

## Experiment 6.2 (Express)

**Name: Prachet Shah**

**SAP: 60009200029**

**Batch: K1**

**Aim:** To implement Express

2. Making API end points using Express.

### **Theory:**

First, initialize a new Node.js project:

```
$ npm init
```

Fill the requested information to requirements. Fields like the name are much more relevant for publishing applications to the Node Package Manager, amongst other fields.

Alternatively, use the default settings by adding the -y flag to the call:

```
$ npm init -y
```

A project with a package.json file is a json file that contains all the relevant metadata on your project and will look something along these lines by default:

```
{
  "name": "app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"
  },
  "author": "",
  "license": "ISC",
  "keywords": [],
  "description": ""
}
```

One can put any number of commands besides node index.js as the start script and when you run npm start - they'll all run:

```
$ test npm start
```

```
> app@1.0.0 start /Users/david/Desktop/app
```

```
> node index.js
```

## Install Express!

```
$ npm install --save express
```

A new file is created in the directory, alongside a `node_modules` directory. The `package-lock.json` file keeps track of dependencies and contains their versions and names:

```
{
  "name": "app",
  "version": "1.0.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "accepts": {
      "version": "1.3.7",
      "resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.7.tgz",
      "integrity": "sha512",
      "requires": {
        "mime-types": "~2.1.24",
        "negotiator": "0.6.2"
      }
    },
    ...
  },
}
```

The `node_modules` directory actually hosts the code of the dependencies, and can get quite large very quickly.

## Creating a Simple Endpoint

Start building a simple "Hello World" app. It'll have a single simple endpoint that just returns a message as a response to request to get the home page.

First, create a file called `hello-world.js`:

```
$ nano hello-world.js
```

Then, import the Express framework within it:

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

Next, instantiate the Express app:

```
const app = express();
```

And set port:

```
const port = 3000;
```

The port will be used a bit later, when the app listens to requests. These three lines are boilerplate

Now, create a simple GET endpoint right beneath the boilerplate. When a user hits the endpoint with a GET request, the message "Hello World, from express" will be returned. Set it to be on the home page, so the URL for the endpoint is /:

```
app.get('/', (req, res) => {  
  res.send('Hello World, from express');  
});
```

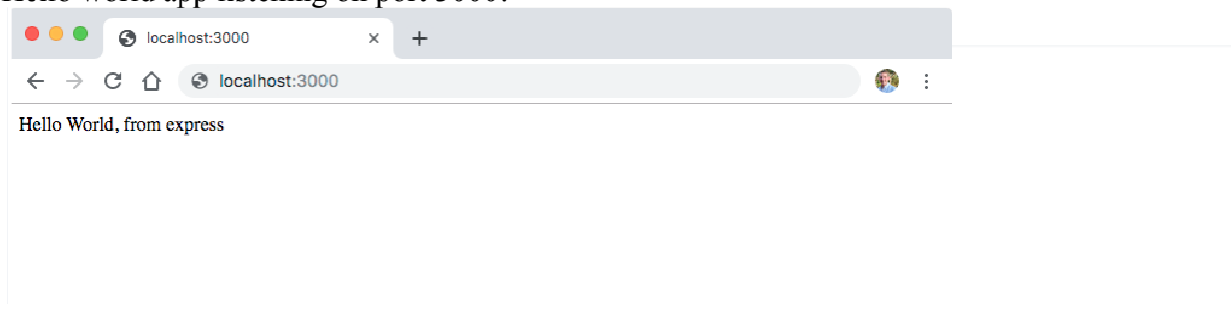
At this point, start clients:

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

Run the application and visit the only endpoint via browser:

```
$ node hello-world.js
```

Hello world app listening on port 3000!



This is technically a working API! Some common middleware that'll be useful for creating some more useful endpoints.

## Express Middleware

As mentioned above - ExpressJS is a simple HTTP server and it does not come with a lot of features out of the box. Middleware act almost like extensions for the Express server and provide additional functionalities in the "middle" of a request. Many third-party extensions likemorgan for logging, multer for handling file uploads, are used routinely. For now, to get started, install a middleware called body-parser, which helps us decode the body from an HTTP request:

```
$ npm install --save body-parser
```

It parses the body of the request and reacts to it accordingly.

Since calling the API from different locations by hitting endpoints in the browser. One can also have to install the CORS middleware.

Start installing the middleware and configure it without being familiar with cross-origin resource sharing:

```
$ npm install --save cors
```

## Building a REST API with Node and Express

## Adding Books

Start building app. Create a new file called book-api.js:

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

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

const cors = require('cors');

const app = express();

const port = 3000;

let books = [];

app.use(cors());

// Configuring body parser middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.post('/book', (req, res) => {

  });

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

One can configure body-parser by importing it and passing it to the app.use method, which enables it as middleware to the Express app instance.

Use the books array to store collection of books, simulating a database.

There are a few types of HTTP request body types. For an example, application/x-www-form-urlencoded is the default body type for forms, whereas application/json is something to use when requesting a resource using jQuery or backend REST client.

What the body-parser middleware will be doing is grabbing the HTTP body, decoding the information, and appending it to the req.body. From there, easily retrieve the information from the form, a book's information.

Inside the app.post method let's add the book to the book array:

```
app.post('/book', (req, res) => {

  const book = req.body;

  // Output the book to the console for debugging
  console.log(book);

  books.push(book);

  res.send('Book is added to the database');

});
```

Now, create a simple HTML form with the fields: ISBN, title, author, published date, publisher, and number of pages in a new file, say new-book.html.

sending the data to the API using this HTML form's action attribute:

```
<div class="container">

  <hr>

  <h1>Create New Book</h1>

  <hr>


  <form action="http://localhost:3000/book" method="POST">
    <div class="form-group">
      <label for="ISBN">ISBN</label>
      <input class="form-control" name="isbn">
    </div>


    <div class="form-group">
      <label for="Title">Title</label>
      <input class="form-control" name="title">
    </div>


    <!--Other fields-->

    <button type="submit" class="btn btn-primary">Submit</button>
  </form>
</div>
```

In this `<form>` tag's attribute corresponds to endpoint and the information send with the submit button is the information method parses and adds to the array.

Output after opening the page:

### Create New Book

ISBN

9781593275846

Title

Eloquent JavaScript, Second Edition

Author

Marijn Haverbeke

Published Date

12/14/2014

Publisher

No Starch Press

Number Of Pages

472

Submit

On Clicking "Submit", greeted with the applications `console.log(book)` statement:

```
{ isbn: '9781593275846',  
  title: 'Eloquent JavaScript, Second Edition',  
  author: 'Marijn Haverbeke',  
  publish_date: '2014-12-14',  
  publisher: 'No Starch Press',  
  numOfPages: '472' }
```

### Getting All Books

Now let's create an endpoint to get all the books from the API:

```
app.get('/books', (req, res) => {  
  res.json(books);  
});
```

Restart the server. If the server is already running press `Ctrl + C` to stop it first. Add some books and open `http://localhost:3000/books` in browser. You should see a JSON response with all the books that you've added.

Now let's create an HTML page to display these books in a user-friendly way.

This time around, create two files - `book-list.html` which will be used as a template and a `book-list.js` file which will hold the logic to updating/deleting books and displaying them on the page:

### Lab Assignments to complete in this session

1. To Make API end points using Express.

### Code:

`book-api.js`

```
// Importing Libraries  
const express = require("express");  
const bodyParser = require("body-parser");  
const cors = require("cors");
```

```

const app = express();

const port = 3000;

let books = [];

app.use(cors());

// Configuration of parser
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.post("/book", (req, res) => {
  const book = req.body;
  console.log(book);
  books.push(book);
  //res.status(200).send("Book is added to database");
  console.log("Book Added");
});

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

```

home.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" />
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.
css" rel="stylesheet"
integrity="sha384-0evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjfhXaWutUpFmBp4vmV
or" crossorigin="anonymous">
  </head>
  <title>Express Exp 6.2</title>
</head>
<body>
  <div class="container">
    <hr />
    <h1>Create New Book</h1>
    <hr />

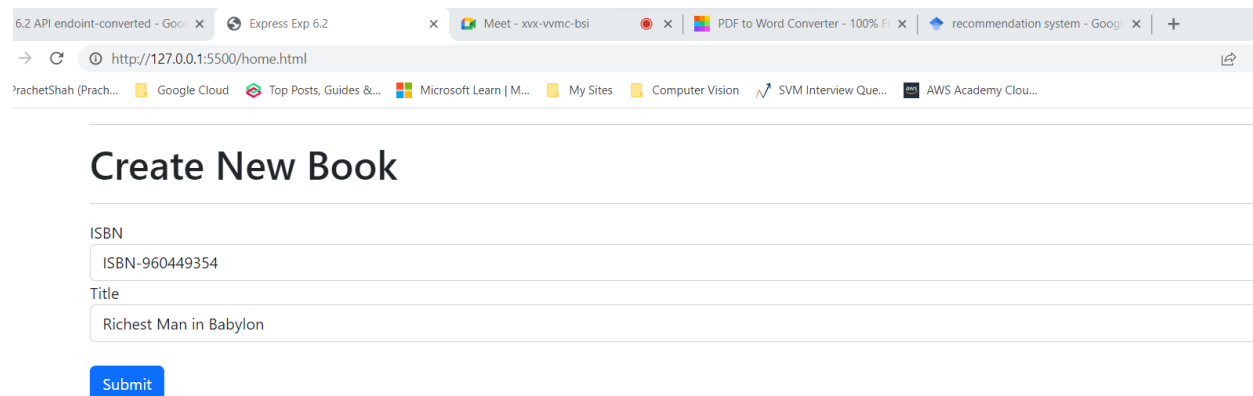
```

```

<form action="http://localhost:3000/book" method="POST">
  <div class="form-group">
    <label for="ISBN">ISBN</label>
    <input class="form-control" name="isbn" />
  </div>
  <div class="form-group">
    <label for="Title">Title</label>
    <input class="form-control" name="title" />
  </div>
  <!--Other fields-->
  <br>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
</div>
</body>
</html>

```

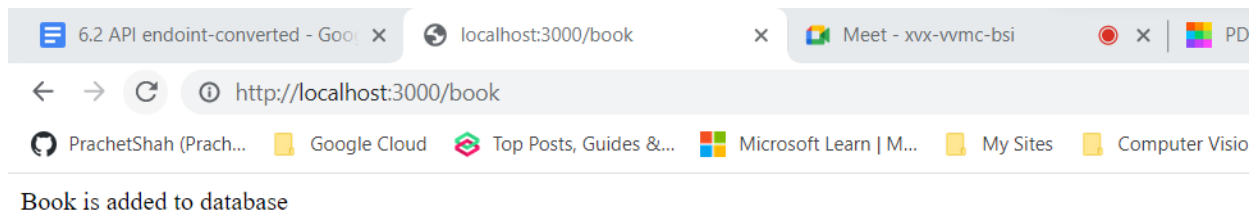
Outputs;



The screenshot shows a web browser window with the address bar displaying 'http://127.0.0.1:5500/home.html'. The browser's tab bar includes several open tabs, and the bookmarks bar shows various links. The main content area features a heading 'Create New Book' followed by a form. The form contains two input fields: 'ISBN' with the text 'ISBN-960449354' and 'Title' with the text 'Richest Man in Babylon'. Below these fields is a blue button labeled 'Submit'.

After pressing Submit:





Console:

```
PS C:\Users\prach\Desktop\DJSC\WE\Exp6.2> node book-api
Server listening on port 3000
[Object: null prototype] {
  isbn: 'ISBN-960449354',
  title: 'Richest Man in Babylon'
}
Book Added
█
```