FS II Notes - Day 7 (Oct 15, 2019)

Server-Side Javascript Programming - ExpressJS

Server Development Template Setup

- Verify NodeJS installation and NPM installation
- Project set up
  - NPM
  - Project Initialization

```
npm init
```

External Information Link:

TASK:

Demonstrate management of development environment for client/server testing setup. Demonstrate file management for project testing and demonstration.

- Verify that NodeJS and NPM are installed and functional within your working environment
- Create a development folder that will be dedicated to ExpressJS server-side code exercises named “js_express” - This is our “Development Exercise Root Repository” (DERR) for ExpressJS server-side javascript
- Create the following within the DERR:

0_app/client/index.html

0_app/client/js

0_app/client/css

0_app/server/

- Initialize server/ as a NPM project

Environment Testing

- ExpressJS server

You can start Express from the folder where the file is located using NodeJS:

```
> node appFilename.js
```

```
const express = require('express')
const appServer = express()
const port = 999

appServer('/', (req, res) => res.send('Hello World'))
appServer(port, () => console.log(`Ok on port ${port}`))
```

External Information Link:

https://expressjs.com/en/starter/hello-world.html

TASK:

Demonstrate simple client/server testing setup. Demonstrate dependency installation and file management for project development.

- Create a project folder named “1_express” using your template folder. Initialize this project with NPM and then install Express into this folder as a dependency.
- Using the sample code above to start (the above code is broken), create a working javascript test file for testing different ports.
- JS error handling should always be used

Express Application Generator

- Installing Express Generator

```
> npx express-generator
```

- Skeleton Application

```
> express testAppName
```

```
const port = 999;
const express = require('express')
const appServer = express();

appServer('/', (req, res) => {
  res.send('Hello World')
});

appServer(port, () => {
  console.log('ok on ' + port)
});
```

External Information Link:

https://blog.npmjs.org/post/162869356040/introducing-npx-an-npm-package-runner

https://expressjs.com/en/starter/generator.html

TASK:

Demonstrate installation of dependencies. Demonstrate file management for project testing.

- Create a project folder named “2_generator” using your template folder. Initialize this project with NPM and then install Express into this folder as a dependency.
- Create a skeleton application using the application generator

Routing

- Routing is handling endpoint requests from clients

```
appServer(endpointPath, function (req, res) {
  res.send('Hello World')
})
```

- Request Methods
  - GET, POST, HEAD, etc.

External Information Links:

https://expressjs.com/en/starter/basic-routing.html

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

TASK:

Demonstrate understanding of basic client/server setup and request types.

- Create a project folder named “3_methods” using your template folder. Initialize this project with NPM and then install Express into this folder as a dependency.
- Using proper file handling create a response handler that returns they type of request made from a client-side HTML file. The responder should recognize at least three different endpoints that have been specified within the server-side code.

Serving Files

```
appServer.use(express.static('public'))
appServer.use('/additionalPath', express.static(‘public'))

appServer.static(localPath, [options])
```

External Information Link:

https://expressjs.com/en/starter/static-files.html

Options for express.static - https://expressjs.com/en/4x/api.html#express.static

TASK:

Demonstrate understanding of relative file handling. Demonstrate serving pages based on user input.

- Make a copy of the template folder named “3_serving_files” in the DERR. The folder should be at the same directory level as the template
  - Create a server that listens for requests on port 80 and serves files based on the URL request:
    - The files should be served from the appropriate directories
    - There should be at least three page options for the user
    - The user should be presented with choices for navigation if no URL or query string data is provided (default page)
    - The user should be presented with at least one page where they are asked to enter information into fields
    - The information they enter should be confirmed and displayed back to the user

404 Handling

```
//  This goes at the end
appServer.use(function (req, res, next) {
 res.status(404).send("File Not Found")
})

appServer.get('/routeToRestrictedAccessArea',
  function checkIfAllowed (req, res, next) {
    if (!req.user.isAllowed) {  //  isAllowed is set earlier
      // User is not allowed, continue to other area, e.g. signup
      next('route')
    } else {
      next()
    }
  }, function getRestrictedContent (req, res, next) {
    RestrictedContent.find(function (err, docForClient) {
      if (err) return next(err)
      res.json(docForClient)
    })
  })
```

External Information Links:

https://expressjs.com/en/guide/error-handling.html

https://expressjs.com/en/starter/faq.html

TASK:

Demonstrate understanding of relative file handling. Demonstrate serving pages based on server-side criteria.

- Make a copy of the template folder named "4_access" in the DERR. The folder should be at the same directory level as the template
  - Create a server that listens for requests on port 80 and serves files based on two types of user: authorized and unauthorized.
  - The client-side files should determine the type of user. The server-side files should determine what is displayed to the user based on input from the client side

FINAL TASK:

- Using your Git account push your js_express DERR to a repo named "FS2_JS_ExpressJS"