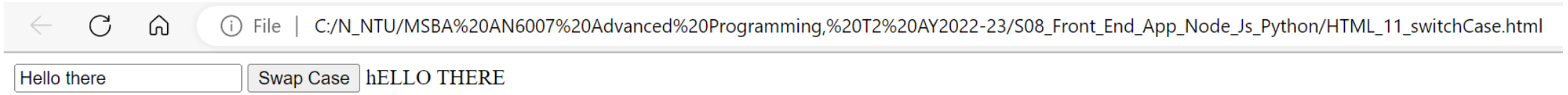
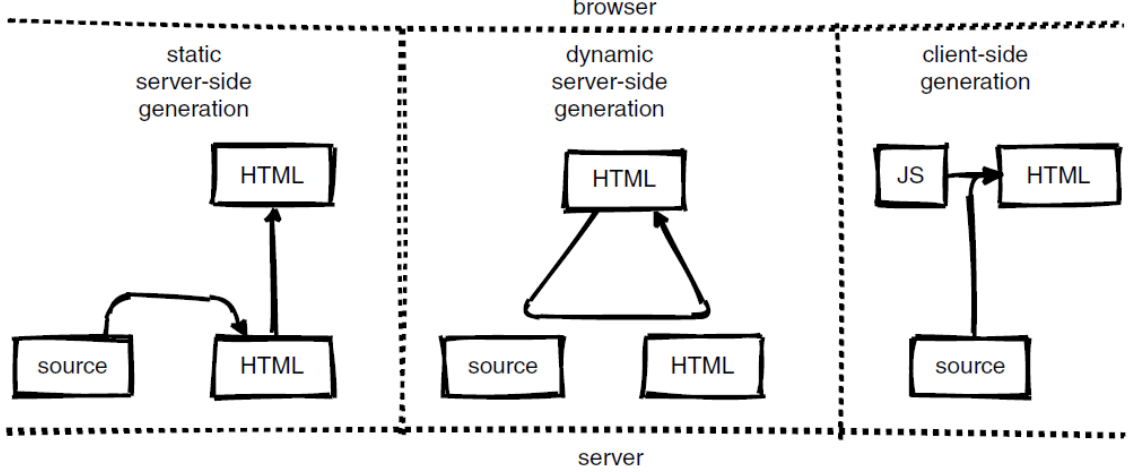
Topic 9 Full Stack
- Program a server

So far ….

- The html we have created all web pages that does not interact with the backend server.
- When launched, you see the site is the html file name



Dynamic Web Pages



- HTML pages are created by typing them in.
- Many pages share a lot of content: navigation menus, contact info, and so on. The nearly universal response was to create a template and embed commands to include other snippets of HTML (like headers) and loop over data structures to create lists and tables. This is called server-side page generation:
  - the HTML is generated on the server, and it was popular because that's where the data was, and that was the only place complex code could be run.

- As browsers and JavaScript became more powerful, the balance shifted toward client-side page generation. I
  - the browser fetches data from one or more servers and feeds that data to a JavaScript library that generates HTML in the browser for display.
  - This allows the client to decide how best to render data, which is increasingly important as phones and tablets take over from desktop and laptop computers.
  - It also moves the computational burden off the server and onto the client device, which lowers the cost of providing data.

Web servers

Development Server - Flask

```
from flask import Flask, jsonify
app = Flask(__name__)
visitors = 0
@app.route('/')
def index():
    global visitors
    visitors+=1
    return f"I am glad to experience flask in MSBA! you are {visitors} visitor"


if __name__ == "__main__":
    app.run(host='localhost', port=5000, debug=False)
```

Set up an HTTP Server by Node.js

```
let http = require('http');
let server = http.createServer(
    function (request, response) {
        // respond to the request
        response.setHeader("Content-Type", "text/plain");
        response.end('Nodejs Http Server at ' + request.url + '\n');
    }
);
server.listen(8080);  // node never stops, mostly waits
```

http1.js

- Use node http1 to activate the server
- Open browser then enter http://localhost:8080/?

HTTP with URL parameters

```
let http = require('http');
let url = require('url');
let server = http.createServer(
    function (request, response) {
        let parsedUrl = url.parse(request.url, true);
        response.setHeader("Content-Type", "text/plain");
        response.end(
          'Hello ' + parsedUrl.query.fname +
          ' ' + parsedUrl.query.lname + '!\n');
    }
);
server.listen(8080);
```

http2.js

node http2 to activate the server

Open browser then enter

http://localhost:8080/?lname=John&fname=Tan

HTTP with paths

```
let http = require('http');
let url = require('url');
let server = http.createServer(
    function (request, response) {
        let parsedUrl = url.parse(request.url, true);
        if (parsedUrl.pathname == '/hello') {
            response.setHeader("Content-Type", "text/plain");
            response.end('We are MSBAians\n');
        } else {
            response.statusCode = 404;
            response.end('Not found!\n');
        }
    }
);
server.listen(8080);
```

http3.js

node http3 to activate the server

Open browser then enter

http://localhost:8080/hello

Npm

Modules → Packages

- Programs are split into *modules*
- Modules can be *packaged* for sending around

Hence: *packages*

Packages

A Package contains …

- Software
  - source code and/or
  - executable binaries
- Metadata about the software:
  - name
  - description
  - version
  - license
  - dependencies

… thus **`package.json`**

A Package Manager …

- Maintains
  - a searchable DB of available packages
  - an archive of package versions
- Enables programmatic discovery & installation of packages
  - Facilitates trust (e.g. usage stats, repo visibility)
- Handles dependencies on behalf of the user
- Advises of security vulnerabilities (and fixes)
- Manages removal of packages
- Avoids manual installation, configuration & build
- Helps standardize project development and runtime environments

… thus **Node Package Manager (npm)**

`npm init` - sets up your project to use npm

- `npm init`
- asks you questions to create your `package.json` file
- this can be manually edited later

`package.json` – metadata

Created in the current directory via: `npm init`

Most important properties in `package.json`:

1. **`name`** - lowercase, will be used in URL and as a directory name
2. **`author`**, **`contributors`** for credits
3. **`description`**, **`keywords`** for search
4. **`bugs`** for users
5. **`license`** for developers
6. … and more on further slides

more information at https://docs.npmjs.com/files/package.json

package.json – dependencies

a package may *depend on* other packages

- dependencies for *using* the package
- devDependencies for *developing* the package

package.json – scripts

main predefined scripts: `test, start, stop, restart`

they will be run with these commands:

- `npm test`
- `npm start` (will try to start `server.js` if `start` script is not specified)
- `npm stop`
- `npm restart` (will do `stop` and `start` if `restart` script is not specified)

Node Package Manager (npm)

The main repository of `npm` packages.

Accessible at `npmjs.org` and the command line via **`npm`**.

Installs as part of **`node.js`**.

`npm install` (shortcut `npm i`)

`npm i` – without params it installs all `dependencies`

and `devDependencies` of current package

- they go into **`node_modules`**
- … where you can inspect and tweak them
- `dependencies` (but not `devDependencies`) of dependencies are also installed

`npm i` *`package`* – installs package locally

and adds a dependency in `package.json`

`npm i` *`package`* `--save-dev` – installs package locally

and adds a dev dependency in `package.json`

`(sudo) npm i -g package` – installs package globally

Nodejs : Creating a package.json file

To create a `package.json` file with values that you supply, use the `npm init` command.

1. On the command line, navigate to the root directory of your package.

```
cd /path/to/package
```

2. Run the following command:

```
npm init
```

3. Answer the questions in the command line questionnaire.

```
> npm init --yes
Wrote to /home/monatheoctocat/my_package/package.json:

{
  "name": "my_package",
  "description": "",
  "version": "1.0.0",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/monatheoctocat/my_package.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/monatheoctocat/my_package/issues"
  },
  "homepage": "https://github.com/monatheoctocat/my_package"
}
```


Creating a package.json file | npm Docs (npmjs.com)

Creating Node.js modules

In the file, add a function as a property of the `exports` object. This will make the function available to other code:

```
exports.printMsg = function() {
  console.log("This is a message from the demo package");
}
```

Test your module

1. Publish your package to npm:
   - For private packages and unscoped packages, use `npm publish`.
   - For scoped public packages, use `npm publish --access public`
2. On the command line, create a new test directory outside of your project directory.

```
mkdir test-directory
```

3. Switch to the new directory:

```
cd /path/to/test-directory
```

4. In the test directory, install your module:

```
npm install <your-module-name>
```

5. In the test directory, create a `test.js` file which requires your module and calls your module as a method.
6. On the command line, run `node test.js`. The message sent to the console.log should appear.

https://youtu.be/3I78ELjTzlQ

Creating Node.js modules | npm Docs (npmjs.com)

About package README files

To help others find your packages on npm and have a good experience using your code in their projects, we recommend including a README file in your package directory. Your README file may include directions for installing, configuring, and using the code in your package, as well as any other information a user may find helpful. The README file will be shown on the package page.

An npm package README file must be in the root-level directory of the package.

Creating and adding a README.md file to a package

1. In a text editor, in your package root directory, create a file called `README.md`.
2. In the `README.md` file, add useful information about your package.
3. Save the `README.md` file.

> **Note:** The file extension `.md` indicates a Markdown file. For more information about Markdown, see the GitHub Guide "Mastering Markdown".

Updating an existing package README file

The README file will only be updated on the package page when you publish a new version of your package. To update your README file:

1. In a text editor, update the contents of the `README.md` file.
2. Save the `README.md` file.

Introducing …

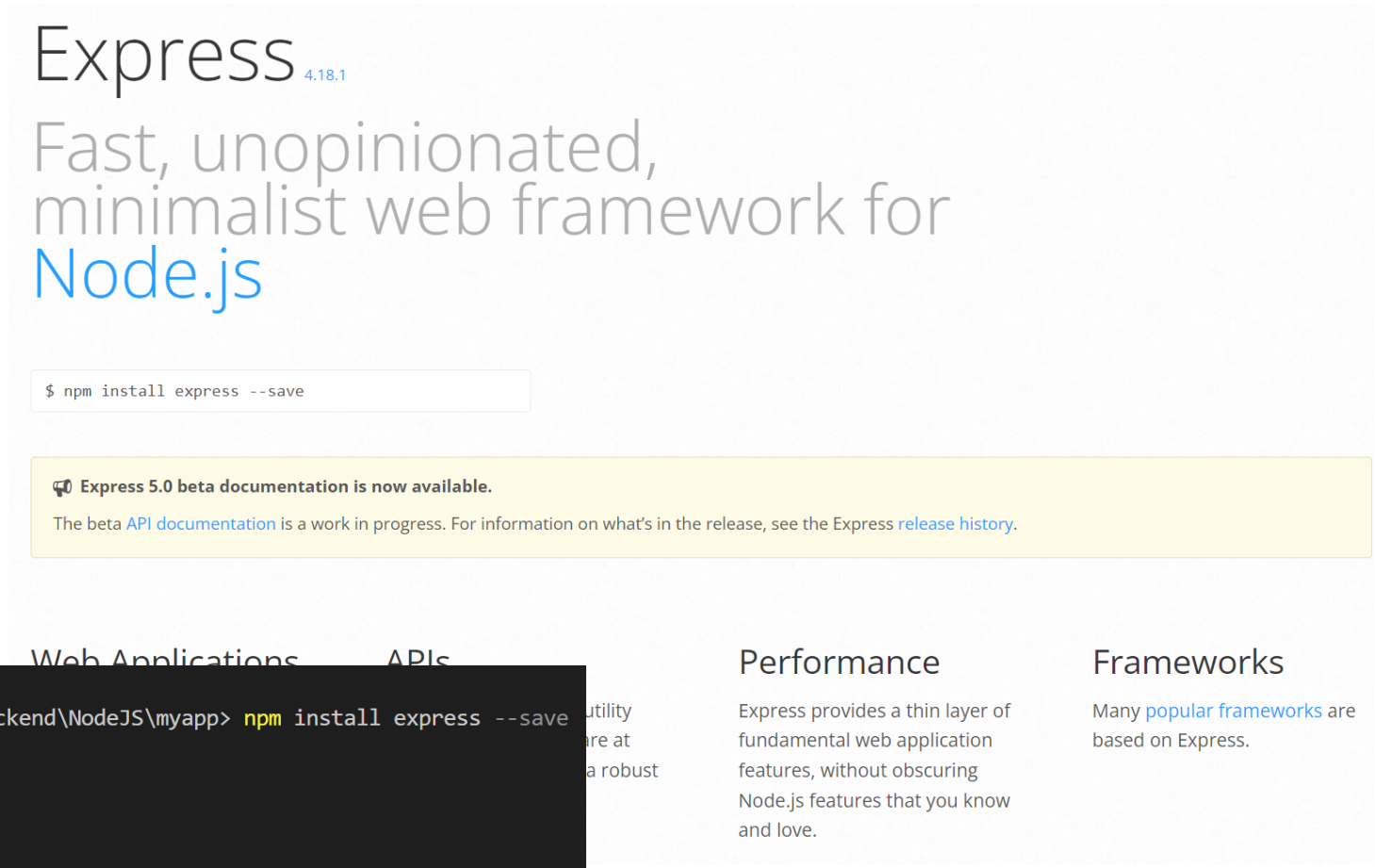
$ npm install express --save

```
Node.js v18.12.1
PS C:\N_NTU\MSBA AN6007 Advanced Programming, T2 AY2022-23\S09_Backend\NodeJS\myapp> npm install express --save

added 57 packages, and audited 58 packages in 2s

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

found 0 vulnerabilities
PS C:\N_NTU\MSBA AN6007 Advanced Programming, T2 AY2022-23\S09_Backend\NodeJS\myapp> node app.js
Example app listening on port 3000
```



[Express - Node.js web application framework (expressjs.com)]

What is Express JS?

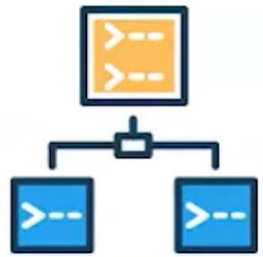
Express is a Node.js framework designed for building APIs, web applications and cross-platform mobile apps, Express is high performance, fast, unopinionated, and lightweight

Features of Express JS

Fast Server-Side Development:
With the help of Node.js features, express can save a lot of time

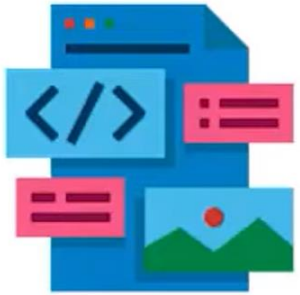
Middleware :
It is a request handler, which have the access to the application's request-response cycle

Routing :
Refers to how an application's endpoints (URIs) respond to client requests



Features of Express JS

Templating :
Creates a html template file with less code and render HTML Pages



Debugging :
Express makes it easier as it identifies the exact part where bugs are

Express in action

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

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

Running Locally

- First create a directory named myap
- Open Terminal, change directory to myapp
- run npm init.
- Then install express as a dependency,

  npm install express
- In the myapp directory, create a file named app.js and copy in the codes provided on the left.
- The req (request) and res (response) are the exact same objects that Node provides
- Run the app with the following command:

  node app.js

Express "Hello World" example (expressjs.com)

- Then, load http://localhost:3000/ in a browser to see the output.

install express as a dependency on visual studio code

- https://code.visualstudio.com/docs/nodejs/nodejs-tutorial

- npm install -g express-generator

- npm install --save express

- npm install -g nodemon

Serving static files in Express

- To serve static files such as images, CSS files, and JavaScript files, use the express.static built-in middleware function in Express.

```
express.static(root, [options])
```

  - The root argument specifies the root directory from which to serve static assets. For more information on the options argument, see express.static.
  - For example, use the following code to serve images, CSS files, and JavaScript files in a directory named public:
  - app.use(express.static('public'))
  - Now, you can load the files that are in the public directory:
    - http://localhost:3000/images/kitten.jpg http://localhost:3000/css/style.css http://localhost:3000/js/app.js http://localhost:3000/images/bg.png http://localhost:3000/hello.html
- Express looks up the files relative to the static directory, so the name of the static directory is not part of the URL.
- To use multiple static assets directories, call the express.static middleware function multiple times:

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

Serving static files in Express (expressjs.com)

list of examples using Express.

- auth - Authentication with login and password
- content-negotiation - HTTP content negotiation
- cookie-sessions - Working with cookie-based sessions
- cookies - Working with cookies
- downloads - Transferring files to client
- ejs - Working with Embedded JavaScript templating (ejs)
- error-pages - Creating error pages
- error - Working with error middleware
- hello-world - Simple request handler
- markdown - Markdown as template engine
- multi-router - Working with multiple Express routers
- multipart - Accepting multipart-encoded forms
- mvc - MVC-style controllers
- online - Tracking online user activity with online and redis packages
- params - Working with route parameters
- resource - Multiple HTTP operations on the same resource
- route-map - Organizing routes using a map
- route-middleware - Working with route middleware
- route-separation - Organizing routes per each resource
- search - Search API
- session - User sessions
- static-files - Serving static files
- vhost - Working with virtual hosts
- view-constructor - Rendering views dynamically
- view-locals - Saving data in request object between middleware calls
- web-service - Simple API service

Web services

- Since the invention of WWW, various web technologies like RPC or SOAP were used to create and implement web services.
- But these technologies used heavy definitions for handling any communication task.
- REST was developed which helped in reducing the complexities and provided an architectural style in order to design the network-based application.

REST API

What is REST? - YouTube

- REST or **RE**ST**ful** stands for **RE**presentational **S**tate **T**ransfer. It is an architectural style as well as an approach for communications purposes that is often used in various web services development. In simpler terms, it is an application program interface (API) that makes use of the HTTP requests to GET, PUT, POST and DELETE the data over WWW.

- REST architectural style helps in leveraging the lesser use of bandwidth which makes an application more suitable for the internet. It is often regarded as the "***language of the internet***". It is completely based on the resources where each and every component is regarded as a component and a single resource is accessible through a common interface using the standard HTTP method.

- REST API breaks down a transaction in order to create small modules. Now, each of these modules is used to address a specific part of the transaction. This approach provides more flexibility but requires a lot of effort to be built from the very scratch.

How to Build REST API with Node.js from Scratch | Edureka

REST API

How The Backend Works - YouTube

What is REST? - YouTube

REST-based architecture

- **GET** – Provides read-only access to a resource.
- **PUT** – Creates a new resource.
- **DELETE** – Removes a resource.
- **POST** – Updates an existing resource or creates a new resource.

Routing

- *Routing* refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
- Each route can have one or more handler functions, which are executed when the route is matched.
- Route definition takes the following structure:
  - app.METHOD(PATH, HANDLER)
    - app is an instance of express.
    - METHOD is an HTTP request method, in lowercase.
    - PATH is a path on the server.
    - HANDLER is the function executed when the route is matched.

The following examples illustrate defining simple routes.

Respond with `Hello World!` on the homepage:

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

Respond to POST request on the root route (`/`), the application's home page:

```
app.post('/', (req, res) => {
  res.send('Got a POST request')
})
```

Respond to a PUT request to the `/user` route:

```
app.put('/user', (req, res) => {
  res.send('Got a PUT request at /user')
})
```

Respond to a DELETE request to the `/user` route:

```
app.delete('/user', (req, res) => {
  res.send('Got a DELETE request at /user')
})
```

Express basic routing (expressjs.com)

Principles of REST

1. Stateless
   Requests sent from a client to the server contains all the necessary information that is required to completely understand it. It can be a part of the URI, query-string parameters, body, or even headers. The URI is used for uniquely identifying the resource and the body holds the state of the requesting resource. Once the processing is done by the server, an appropriate response is sent back to the client through headers, status or response body.
2. Client-Server
   It has a uniform interface that separates the clients from the servers. Separating the concerns helps in improving the user interface's portability across multiple platforms as well as enhance the scalability of the server components.
3. Uniform Interface
   To obtain the uniformity throughout the application, REST has defined four interface constraints which are:
   1. Resource identification
   2. Resource Manipulation using representations
   3. Self-descriptive messages
   4. Hypermedia as the engine of application state
4. Cacheable
   In order to provide a better performance, the applications are often made cacheable. It is done by labeling the response from the server as cacheable or non-cacheable either implicitly or explicitly. If the response is defined as cacheable, then the client cache can reuse the response data for equivalent responses in the future. It also helps in preventing the reuse of the stale data.
5. Layered system
   The layered system architecture allows an application to be more stable by limiting component behavior. This architecture enables load balancing and provides shared caches for promoting scalability. The layered architecture also helps in enhancing the application's security as components in each layer cannot interact beyond the next immediate layer they are in.
6. Code on demand
   Code on Demand is an optional constraint and is used the least. It permits a clients code or applets to be downloaded and extended via the interface to be used within the application. In essence, it simplifies the clients by creating a smart application which doesn't rely on its own code structure.

Building API with Express and MongoDB

- Build A REST API With Node.js, Express, & MongoDB - Quick - Bing video