# NODE.JS

# Dynamic Web Pages Server Side

- There are a number of technologies that allow us to create web pages on the fly on the server.  Usually, there is some data-base that will back up the pages
- Example:  You do a google search and get back a constructed web-page that has your search results.

# Node Architecture

- Event loop based server provides responses for requests.
- The response can be html or other types like plain text.
- Runs on a single process so avoids some concurrency issues.  Does not spin off a thread for each request.
- I/O operations are non-blocking (asynchronous).  When the I/O operation finishes, continue with the code after the I/O.
- Good scalability

# Node Architecture

- Uses a familiar language (JavaScript)
- Runs on the V8 JavaScript engine from Chrome
- Not on a browser so no DOM or any objects provided by the browser
- The developer controls the APIs that will be used.

# Installing NodeJS/Express

- The official documentation and download
- The long term stable (LTS) version is fine for our purposes.
- You will want all the options.
- You will want to select installing the tools for the native modules, which will trigger a second installation script after the first completes.
  - *This also installs chocolaty*

# A Simple Server

- We want to build a program that listens on a given port.
  - *List of common ports and what listens there.*
- Requests come in and it returns a response.
- The response will be plain text.
- Uses string interpolation with ${expression} inside of a back-tick marked string.
  - *Ex:* `` `My friend ${friend.name} is ${friend.age*12} months old` ``

# A Simple Server

```
const http = require('http’)  //Pull in a useful node package
                              //Try http.
const hostname = '127.0.0.1’  //Local host
const port = 3001 //Not assigned

const server =
  http.createServer( //Creates the response loop
      (req,res)=> { //Anonymous function to handle the request
        res.statusCode = 200 //code for OK
        res.setHeader('Content-Type', 'text/plain') //Set the mime type
        res.end('Hello World') //Close the response and provide content
      } //No return needed, we modified the res object we got
  )

server.listen(port, hostname, () => { //Start the server
console.log(`Server running at http://${hostname}:${port}/`)
         //Log the start
})
```

# Notice:

- This uses template literals
  - *Uses backtick pairs*
  - *Allows us to do string interpolation where an expression is evaluated and slotted into the string. ${expression}.*
  - *A little bit cleaner than doing concatenation.*
  - *Reference*
- Local Host (loop back) – an IP address that is associated internally with our computer.
  - *Useful for testing purposes.*
  - *Finding  our machines IP address can be tricky*

# Running the server

- Assuming it is saved in the file app.js
- The .js file has code but is not HTML, so no tags.
- It is running in node.
- Control-C to stop it

In a command line.

```
node app.js
```

Or to start up a read-eval-print loop do

```
node
```

# A Less Simple Server

```
const http = require('http') //Pull in a useful node package
const hostname = process.env.hostname || '127.0.0.1' //get our ip address
from the environment
const port = process.env.port || 3001 //and the port

const server =
  http.createServer( //Creates the response loop
    (req,res)=> { //Anonymous function to handle the request
    res.statusCode = 200 //code for OK
    res.setHeader('Content-Type', 'text/html') //Set the mime type HTML

    res.write('<html> <head> <title> Served </title> </head>')
    res.write('<body>')
    res.write('Content \n')
    res.write('More content \n')
    res.write('Hello World')
    res.end('</body></html>')
    //Close the response
}
)

server.listen(port, hostname, () => { //Start the server
console.log(`Server running at http://${hostname}:${port}/`) //Log the
start
})
```

# NPM

- Node Package Manager
- Used to manage more complicated projects

# JSON

- JavaScript Object Notation
- XML and JSON are popular string based representations of objects.
  - *Used to send structured information from a source to a receiver.*
  - *Serialization/Deserialization*
- Mostly already familiar to you.
  - *Use {} for an object and [] for an array*
- Uses string as the name/key for a mapping in an object.
- Functions are not one of the supported data types.

# package.json

- Manifest and meta data for a project
- Uses JSON
- Gives dependencies
  - *Allows one to split into dependencies that are used in production and development.*
- Reference

# Generator for package.json

- Name
- version
- Description
- Entry point
- Test command
- Git repo
- Keywords
- Author
- License

In a command line.
In the directory where your project lives.

```
npm init
```

# Express

- Framework living on top of node.
- Makes it easy to specify the endpoints (legal path) for your app. (Node can do endpoints as well if we use the url property of the request, but this is cleaner.)
- Supports a number of templating engines that can dynamically create content.
- Relatively easy to respond with a resource.

# Installing express

In a command line.

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

The save will add express to the dependencies in the package.json.

# An Express Server (ex1.js)

```
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 at
http://localhost:${port}`)
})
```

Only accepts GET requests. All others generate a 404 response.

Create directory ex1app
Change to it
Add file ex1.js

```
npm init
npm install express -save
node ex1.js
```

Browse at `http://localhost:3000/`

# Generator for Express Apps

- It is more common to use the express generator to create the skeleton of an app that you can flesh out to meet your particular needs.
- You tell it the root directory and it will create that directory and the underlying app directories.
- You choose which view engine you want to use.
  - *ejs (encapsulated javascript)*
  - *hbs (handlebars)*
  - *pug*
  - *hogan*
  - *jade*
  - *vash*
  - *Twig*
- Defaults to jade for now

In a command line.
In the directory where your
app named ex2app is going to live.

```
npx express-generator –view=pug
npm init
npm install
```

```
DEBUG=ex2app:* npm start
```
(unix version)

# Generator for Express Apps

- You choose a css engine you want to use.
  - *less*
  - *stylus*
  - *compass*
  - *sass*

- Default is css

# Out of Date/Audit

- When you do the install, you may find that certain dependencies are out of date and/or have vulnerabilities.
- npm outdated – which files are out of date
- npm audit – which dependencies have vulnerabilities.
- Fix by changing the dependency, installing the new version or through audit. Audit will not update over major version changes ("breaking changes") unless you use the –force flag.

```
npm audit fix
npm install debug@4.3.0 pug@latest
```

- Circumstances may dictate using an older version of a package. Easier to update early in the project life-cycle.

# Out of Date

npm outdated
npm install debug@4.3.0 pug@latest

| Package | Current | Wanted | Latest | Location |
|---|---|---|---|---|
| debug | 2.6.9 | 2.6.9 | 4.3.1 | goapp |
| express | 4.16.4 | 4.16.4 | 4.17.1 | goapp |
| http-errors | 1.6.3 | 1.6.3 | 1.8.0 | goapp |
| morgan | 1.9.1 | 1.9.1 | 1.10.0 | goapp |
| pug | 2.0.0-beta11 | 2.0.0-beta11 | 3.0.2 | goapp |

breaking change

minor change

```
"dependencies": {
"cookie-parser": "~1.4.4",
"debug": "~2.6.9",
"express": "~4.16.1",
"http-errors": "~1.6.3",
"morgan": "~1.9.1",
"pug": "2.0.0-beta11"
}
```

```
"dependencies": {
"cookie-parser": "~1.4.4",
"debug": ”^4.3.1",
"express": "~4.16.1",
"http-errors": "~1.6.3",
"morgan": "~1.9.1",
"pug": ^3.0.2"
}
```

# Out of Date

npm outdated

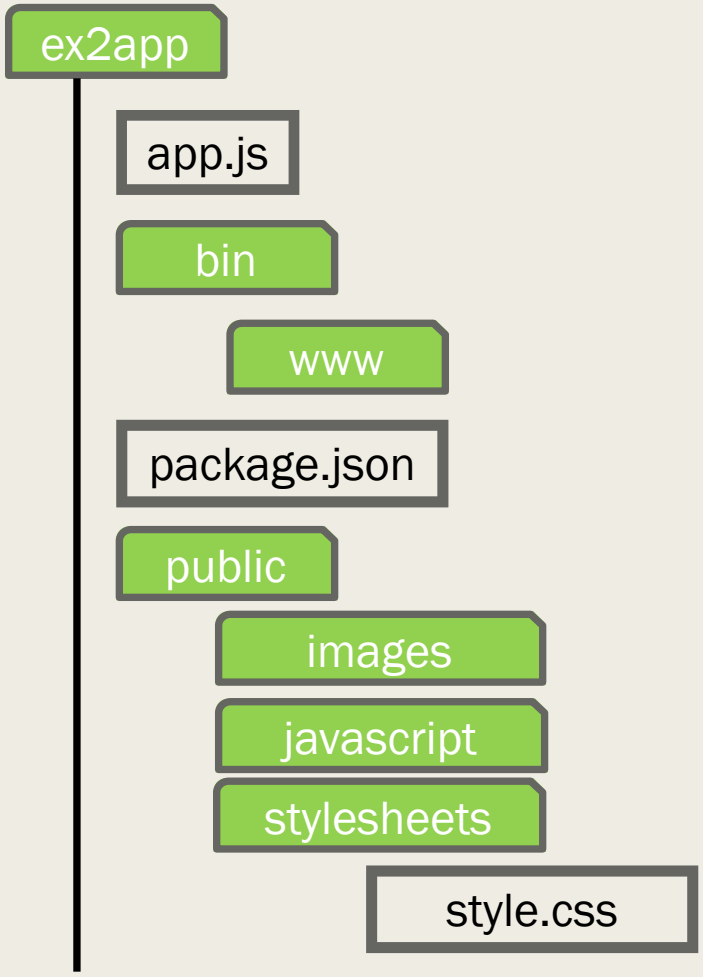
| Package | Current | Wanted | Latest | Location |
|---|---|---|---|---|
| debug | 4.3.0 | 4.3.1 | 4.3.1 | goapp |
| express | 4.16.4 | 4.16.4 | 4.17.1 | goapp |
| http-errors | 1.6.3 | 1.6.3 | 1.8.0 | goapp |
| morgan | 1.9.1 | 1.9.1 | 1.10.0 | goapp |

```
"dependencies": {
"cookie-parser": "~1.4.4",
"debug": ”^.4.3.1",
"express": "~4.16.1",
"http-errors": "~1.6.3",
"morgan": "~1.9.1",
"pug": ”^3.0.2"
}
```

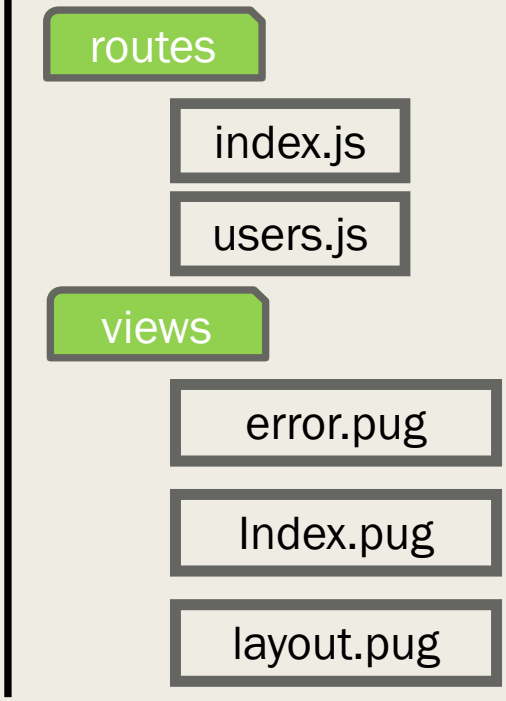
Which version should I use…
~ is any version after this one that only changes last number (patches)
^ is any version after this one that only changes in last two numbers (minor and patch)

We can directly change the package.json and then do npm install.

# Express App Structure - PUG (ex2app)



# Express App Structure (ex2app)



# Integration with Git

- We are going to want to put our project into a git repo and there is generated content that we don't want under version control.
- Search ".gitignore express" and look at the github source:

https://github.com/expressjs/express/blob/master/.gitignore

- Copy to the project

# .gitignore

<<OS specific patterns not shown>>

```
# npm
node_modules
package-lock.json
*.log
*.gz

# Coveralls
coverage

# Benchmarking
benchmarks/graphs
```

# Making a Git repo

- Make the local repo

```
git init
git status
```

- Stage everything

```
git add .
```

- Tell it where my upstream repo is (previously created on github

```
git remote add origin https://github.com/Charles-Hoot/node-express-sample-ex2app.git
```

# Making a Git repo

- Lets do a good commit message

  `git commit -m ”Initial repo push to remote”`

- Change the name of the branch for the local repo (master) to match the remote (main)

  `git branch –M main`

- Do the push. First time we need to specify the upstream

  `git push –u origin main`

# Cloning it down

- We don’t see things like node-modules in the remote repo, but it is still in the working directory of the local. Lets see what happens when we clone down the remote.  Change to another directory and then clone.

```
cd ..
git clone https://github.com/Charles-Hoot/node-express-sample-ex2app.git
```

- This creates a new local repo tied to the remote. (Not usually what you want, but think about working on a different computer.  If we try to run this, it won’t work because none of the dependencies have been added to the app. Do an install and you are good to go.

```
npm install
```

# app.js

Lots of packages being pulled in. Express has 50ish dependencies

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
```

Using pug to create the views (pages)

# app.js

Middle ware

```
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
```

Allow access to the public directories

```
app.use(express.static(path.join(__dirname,
'public')));
```

A router expresses the paths I can follow and get to a page/service. This has 2 endpoints

```
app.use('/', indexRouter);
app.use('/users', usersRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});
```

# app.js

```
// error handler
app.use(function(err, req, res, next) {
   // set locals, only providing error in
development
   res.locals.message = err.message;
   res.locals.error = req.app.get('env') ===
'development' ? err : {};

   // render the error page
   res.status(err.status || 500);
   res.render('error');
});

module.exports = app;
```

Will use the error view

# Routes – index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
      res.render('index', { title: 'Express' });
});

module.exports = router;
```

Will use the index view

# View – index.pug

```
extends layout

block content
h1= title
p Welcome to #{title}
```

Specific content using shared layout

# View – layout.pug

```
doctype html
  html
  head
  title= title
  link(rel='stylesheet', href='/stylesheets/style.css')
  body
  block content
```

Template for creating HTML

# Routes – users.js

```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
    res.send('respond with a resource');
});

module.exports = router;
```

No need to build HTML, send the resource.

# View – error.pug

Child

```
extends layout

block content
  h1= message
  h2= error.status
  pre #{error.stack}
```

# nodemon

- A demon that watches for changes in the app source and restarts the server on the fly.
- Can create it globally, but we will do it locally on a particular app.
- To run it locally we get the version that is buried under the node_modules directory. This will automatically use the start script from package.json

node_modules/nodemon/bin/nodemon.js

# Routers – HTTP types

- GET and POST are the most common types
  - *GET – Get data from a specified resource. Can be cached. Can be bookmarked. Stored in history.*
  - *HEAD - Like GET but has no body.  Allows you to check things like length before doing the get.*
  - *POST - Send data to update a specified resource. No cache, bookmark, or history.*
  - *PUT – Like post, but multiple requests with the same PUT give back the same result. Replace the resource with something.*

# Routers – HTTP types

- Specify the type of HTTP request you will respond to:
  - *DELETE – delete the resource*
  - *PATCH - Make partial changes to a specified resource.*
  - *OPTIONS – get the communication options for the target resource*
  - *CONNECT – Start 2-way communication. Open a tunnel.*
  - *TRACE - Do a message loop back from the target resource.*

# Routers

- Express allows us to easily specify an endpoint (URI) and the type of request and how to manage it.
- `app.TYPE(PATH, HANDLER )`
  - *‘*’ as a path will match anything not yet matched.*

- Example:

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

# Routers

- Example:

```
router.delete(‘/user/1', function(req, res,
next) {
      res.send(’Delete for /user/1’)
});
```

# Middleware

- Something we do to every request before it gets routed.
  - *Example: Log every request.*
  - *Example: Authenticate request.*

  - *app.use(logger('dev'));*

# EJS - Views

- If you use –view=ejs in the generator we can get embedded java script. Here is what the views look like. (Compare with pug.)

```
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
  <link rel='stylesheet'
     href='/stylesheets/style.css' />
</head>
<body>
  <h1><%= title %></h1>
  <p>Welcome to <%= title %></p>
</body>
</html>
```