44-563: Unit 12

Developing Web Applications and Services

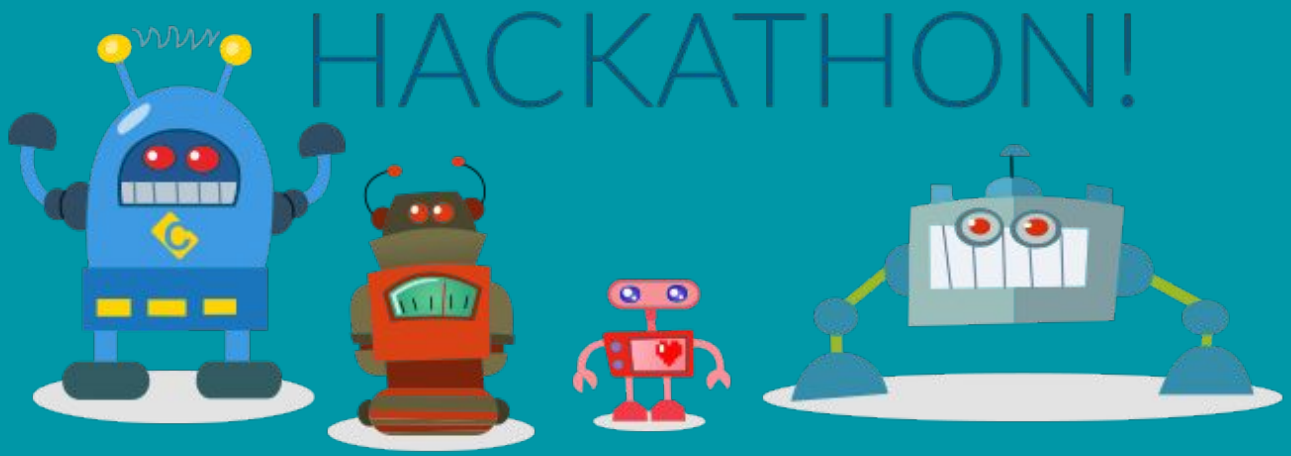
Includes

- 12 architectural areas
- M11 - building a full stack app
- Persistence
- Security
- Deployment

12 architectural areas

1. root folder (.gitignore, README.md, LICENSE.md, package.json, app.js, .env.example)
2. /data/ (estimates.json, users.json)
3. /controllers/ (estimate.js, user.js, contact.js, about.js)
4. /models/ (estimate.js, user.js)
5. /routes/ (index.js)
6. /views (layout.ejs, index.ejs, header.ejs, footer.ejs) & views/estimate (create.ejs, delete.ejs,details.ejs, edit.ejs, index.ejs) [partial_edit.ejs, select.ejs]
7. /utils/ (logger.js, seeder.js)
8. /test/controllers/ (estimateTest.js and userTest.js)
9. /test/models/ (estimateTest.js, userTest.js)
10. /config (ensureAuthenticated.js, passportConfig.js)
11. /views (about.ejs, 404.ejs) & /views/account (forgot.ejs, login.ejs, profile.ejs, reset.ejs, signup.ejs)
12. /public (client-side assets, e.g., css, images, js) + Contact page (all)

M12



http://co-station.com/wp-content/uploads/2017/05/hackathon-graphic.png

M12

- Today, in class, **project teams (2 units)** will implement basic functionality.
- Push and pull often.
- Goal: **working app by end of day**
  - First, we will talk through what needs to be done by each team.
  - Then, we will break to implement it.
- Attend tonight's ACM-G workshop in lieu of the help session
- Work together while enjoying pizza :)

Team 1

Complete initial files in this order (use all members):

1. .**gitignore** (copy from ref - pretty standard for express app) and **package.json** (use ref - change name to cge0x)
2. **app.js** (see slides)
3. **README.md** & **LICENSE.md** (Use Apache 2) and create a **.env** file to hold environment variables (see slide)
4. **/public** (will be used as we add views) - copy css, js, image folders and contents from ref project.

Always pull first.

Make your modifications.

Add any new files and commit locally.

Push to origin.

We can't do anything until your files are in place. :)

Team 1 - app.js (part 1)

```
/**
* @file app.js
* The starting point of the application.
* Express allows us to configure our app and use
* dependency injection to add it to the http server.
*
* The server-side app starts and begins listening for events.
*
*/

// Module dependencies
const express = require('express')
const http = require('http')
const path = require('path')
const engines = require('consolidate')
```

- Require the packages needed for an express app: http & express.
- Require path so we can add paths to our app
- Require consolidate to set up our view engine (ejs) - additional view engines are possible.

Team 1 - app.js (part 2)

```
// app variables
const DEFAULT_PORT = 8089 // make last digit = section num

// create express app ................................
const app = express()

// configure app.settings.............................
app.set('port', process.env.PORT || DEFAULT_PORT)

// set the root view folder
app.set('views', path.join(__dirname, 'views'))

// specify desired view engine
app.set('view engine', 'ejs')
app.engine('ejs', engines.ejs)



// start Express app
app.listen(app.get('port'), () => {
 console.log('App is running at http://localhost:%d in %s mode', app.get('port'), app.get('env'))
 console.log('  Press CTRL-C to stop\n')
})

module.exports = app
```

- Implement a basic express app until more functionality is ready.

Team 1 - .env

```
PORT=8089
NODE_ENV=production
```

- Create a new file ".env" in the root folder.
- Use this to set some basic environment variables

- .env may hold access credentials which are secret.

- Generally, we do not commit **.env** to the repo, but we create and commit a **.env.example** that each user can use to create their personal .env.

Team 1 - .env

```
PORT=8089
NODE_ENV=production
```

- Create a new file ".env" in the root folder.
- Use this to set some basic environment variables

- .env may hold access credentials which are secret.

- Generally, we do not commit **.env** to the repo, but we create and commit a **.env.example** that each user can use to create their personal .env.

Team 1 - public/js

- For performance reasons, make js libraries available locally.
- Download the current, minified versions of:
  - Bootstrap
  - jQuery
  - Modernizr
- If you wish, put external JS in 'vendor' subfolder to keep it separate from our app-specific js.

Team 1 - public/css

- For performance reasons, make css libraries available locally.
- Download the current, minified versions of:
  - Bootstrap
  - Bootstrap-theme
- If you wish, put external CSS in 'vendor' subfolder to keep it separate from our app-specific css.
- Add a main.css for our custom styles.

Team 1 - public/images

- Icons are kept here.
- For now, copy the whole set from the reference project.
- Later, ask a volunteer to design a custom icon and use a favicon generator program from the web to create the versions needed. (Many people love design and would be great at this!)

Team 1 - main.css

```
/* ======   needed for BootStrap menu ====== */

body {
  padding-top: 50px;
  padding-bottom: 20px;
}

.dropdown-submenu {
  position: relative;
}

.dropdown-submenu>.dropdown-menu {
  top: 0;
  left: 100%;
  margin-top: -6px;
  margin-left: -1px;
  -webkit-border-radius: 0 6px 6px 6px;
  -moz-border-radius: 0 6px 6px;
  border-radius: 0 6px 6px 6px;
}

.dropdown-submenu:hover>.dropdown-menu {
  display: block;
}
```

```
.dropdown-submenu>a:after {
  display: block;
  content: " ";
  float: right;
  width: 0;
  height: 0;
  border-color: transparent;
  border-style: solid;
  border-width: 5px 0 5px 5px;
  border-left-color: #ccc;
  margin-top: 5px;
  margin-right: -10px;
}

.dropdown-submenu:hover>a:after {
  border-left-color: #fff;
}

.dropdown-submenu.pull-left {
  float: none;
}

.dropdown-submenu.pull-left>.dropdown-menu {
  left: -100%;
  margin-left: 10px;
  -webkit-border-radius: 6px 0 6px 6px;
  -moz-border-radius: 6px 0 6px 6px;
  border-radius: 6px 0 6px 6px;
}
```

Team 2

Complete initial files in this order:

- Commit **data/estimates.json**.
- Copy entire views/**about** folder from ref app.
- Copy views/**about.ejs** & **about_details.ejs from ref app.**
- More on users.json & views/account when we cover passport & security.

Always pull first.

Make your modifications.

Add any new files and commit locally.

Push to origin.

Team 2 - about.ejs

```
<li class="dropdown">
 <a href="#!" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true"
aria-expanded="false">About
   <span class="caret"></span>
 </a>
 <ul class="dropdown-menu">
   <% include about_details %>
     <li role="separator" class="divider"></li>
     <li class="dropdown-submenu">
       <a href="#">Mentors <span class="caret"></span> </a>
       <ul class="dropdown-menu">
         <li><a href="#">Instructor Alice</a></li>
         <li><a href="#">Instructor Bert</a></li>
         <li><a href="#">Instructor Connie</a></li>
         <li><a href="#">Instructor Doug</a></li>
         <li role="separator" class="divider"></li>
         <li><a href="#">Assistant Sue</a></li>
         <li><a href="#">Assistant Ken</a></li>
         <li><a href="#">Assistant Betty</a></li>
       </ul>
     </li>
 </ul>
</li>
```

- This is part of the nav bar that provides the menu to our about pages.
- Use <% include about_details.ejs %> to insert the team menu information from another file.

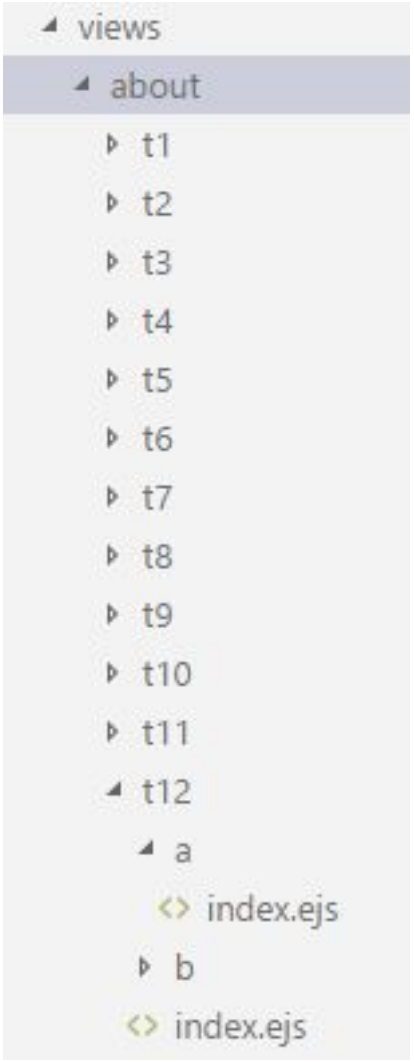
Team 2 - about_details

```
<li class="dropdown-submenu">
 <a href="#">2017 Fall Web Apps Section 01<span class="caret"></span></a>
 <ul class="dropdown-menu">
   <li class="dropdown-submenu">
     <a href="/about/t1">Team 1 <span class="caret"></span></a>
     <ul class="dropdown-menu">
       <li><a href="/about/t1/a">yourname</a></li>
       <li><a href="/about/t1/b">yourname</a></li>
     </ul>
   </li>
   <li class="dropdown-submenu">
     <a href="/about/t2">Team 2 <span class="caret"></span></a>
     <ul class="dropdown-menu">
       <li><a href="/about/t2/a">yourname</a></li>
       <li><a href="/about/t2/b">yourname</a></li>
     </ul>
   </li>


...(repeats for 12 teams (aka "units"))....
```

- This holds the nav bar links to each developer's about page.
- Recommendation: Copy from ref app **about_details.ejs**.
- Customize the heading for your section.
- Needs a menu item for each of the 12 programming units with a sublist for each developer in that unit.
- If desired, consolidate the menu into 6 teams (be careful if you do - many cascading updates will be required)

Team 2 - about_details



- Each unit has their own folder with an index.ejs for the programming unit.
- Within each unit, we have a default of 2 developers (called "a" and "b" for convenience).
- If a unit has less than 2 people, remove all references to "b".
- If a unit has more than 2 people, duplicate all functionality for another developer "c".
- Suggestion: implement your about pages first, so you can help others.

T2 / Everyone - index.ejs

```
<br/>
<br/>
<br/>
<h1>My Name</h1>
<h2>Software Developer</h2>

<br/>
<br/>
```

- Team 2 will create these files initially to mitigate merge conflicts.
- Once available, each developer should complete their own index.ejs (you may add additional pages).
- Here's a simple version - but it would be better to present the about page you've worked on this semester.
- Suggestion: Complete this as soon as Team 2 makes them available - don't wait.

Team 3

Complete work in this order:

- controllers/**about.js** - copy from ref app. This file manages requests for our about pages.
- controllers/**estimate.js**
- controllers/user.js and config are related to security and passport - more on these later.

Always pull first.
Make your modifications.
Add any new files and commit locally.
Push to origin.

Team 3 about.js

```
const express = require('express')
const api = express.Router()

// Specify the handler for each required combination of URI and HTTP verb

// HANDLE VIEW DISPLAY REQUESTS ------------------------------------------

// GET t1
api.get('/t1', (req, res) => {
 console.log(`Handling GET ${req}`)
 res.render('about/t1/index.ejs',
       { title: 'TeamName', layout: 'layout.ejs' })
})
api.get('/t1/a', (req, res) => {
 console.log(`Handling GET ${req}`)
 res.render('about/t1/a/index.ejs',
       { title: 'TeamMember1PutYourNameHere', layout: 'layout.ejs' })
})
api.get('/t1/b', (req, res) => {
 console.log(`Handling GET ${req}`)
 return res.render('about/t1/b/index.ejs',
       { title: 'TeamMember2PutYourNameHere', layout: 'layout.ejs' })
})
// GET t2... through t12 here... and then finally:
module.exports = api
```

- Team 3 will create the controller that handles requests to about pages.
- Do this **early** - all developers need this to implement access to their pages.
- Once you've committed the general version, each team is responsible for customizing the title and content of their team page and individual about pages.

T3/All - estimate.js

```
const express = require('express')
const api = express.Router()
const Model = require('../models/estimate.js')
const LOG = require('../utils/logger.js') // comment out until exists
const find = require('lodash.find')
const remove = require('lodash.remove')
const notfoundstring = 'estimates'

// RESPOND WITH JSON DATA  ----------------------------------------
// GET all JSON
// GET one JSON by ID
// RESPOND WITH VIEWS  --------------------------------------------
// GET to this controller base URI (the default)
// GET create
// GET /delete/:id
// GET /details/:id
// GET one
// HANDLE EXECUTE DATA MODIFICATION REQUESTS -----------------------
// POST new
// POST update
// DELETE id (uses HTML5 form method POST)

module.exports = api
```

- Team 3 will create the controller that handles requests related to estimates.
- Use the comments to assign each method to a programming unit (see your course assistant for their recommendations)
- Add the assigned unit to the handler comment.
- Do this **early** - all developers need this to implement their controller method.

```
/* 10 standard CRUD request handlers per controller

   controllers/estimate.js


2 Respond with JSON:

http://127.0.0.1:8082/estimate/findall      [WORKING]
http://127.0.0.1:8082/estimate/findone/1  [WORKING]

5 Respond with CRUD Views:

http://127.0.0.1:8082/estimate                  [WORKING]
http://127.0.0.1:8082/estimate/create          [WORKING]
http://127.0.0.1:8082/estimate/delete/1       [404]
http://127.0.0.1:8082/estimate/details/1       [404]
http://127.0.0.1:8082/estimate/edit/1          [FOUND]

3 Respond by executing CRUD actions:

http://127.0.0.1:8082/estimate/save       [WORKING]
http://127.0.0.1:8082/estimate/save/1     [WORKING]
http://127.0.0.1:8082/estimate/delete/1   [WORKING]

*/
```

TEAM 3

Maintain this block as a comment in your controller.

Add the [STATUS] for each handler.

First, connect the handler to the response. Then, finalize the response.

Team 4

Complete work in this order:

- models/**estimate.js** - build the outline from the slides. Other teams will commit their parts of the estimate model from the slides as well.
- test/models/estimate.js - add a few unit tests for the estimate model (read up on testing content & feel free to google outside resources - there is no testing in the ref app)
- models/user.js is related to security and passport - more on these later.

Always pull first.

Make your modifications.

Add any new files and commit locally.

Push to origin.

T4/All - estimate.js

```
const mongoose = require('mongoose')  //unit 02 initialize
var Schema = mongoose.Schema

const EstimateSchema = new mongoose.Schema({

 // unit 01

 // unit 03

 // unit 04

 // unit 05

 // unit 06

  // unit 07

  // unit 09

 // unit 11

 })
module.exports = mongoose.model('Estimate', EstimateSchema)
```

- Team 4 commits and pushes the outline for the estimate model.
- Use the comments to assign each part to a programming unit.
- Team 4 - Do this **early** - all developers need your outline first, before they can add their part of the estimate model.
- Units - complete your part of the model. Commit and push. Do this early.

Team 5

Complete work in this order:

- routes/**index.js** - Express uses routing to delegate requests to controllers.
- views/**404**.ejs - create a custom page-not-found page.
- Ask **Team 1** to enable your routing & error functionality in app.js.
- test/controllers/estimate.js - add unit tests for estimate controller

Always pull first.

Make your modifications.

Add any new files and commit locally.

Push to origin.

Team 5 routes/index.js

```
/**
* @index.js - manages all routing
*
* router.get when assigning to a single request
* router.use when deferring to a controller
*
* @requires express
*/

const express = require('express')
const LOG = require('../utils/logger.js')  // comment out until exists
LOG.debug('START routing')   // comment out until exists
const router = express.Router()

// Manage top-level request first
router.get('/', (req, res, next) => {
 LOG.debug('Request to /')  // comment out until exists
 res.render('index.ejs', { title: 'CGE01' }) //  use your sec num
})

// Defer path requests to a particular controller
router.use('/about', require('../controllers/about.js'))
router.use('/estimate, require('../controllers/estimate.js'))

LOG.debug('END routing')
module.exports = router
```

- Team 5 will set up all routing in routes/index.ejs.
- Do this **early** - controllers can't handle requests until they get them.
- Comment out anything that uses utilities until they exist. Once the logger is added and available, remove the comments and enable this functionality.
- See comments for when to use get() & use()

Team 5 views/404.ejs

```
<br>
<br>
<div >
 <h1>404</h1>
 <p> Page Not Found </p>
</div>
```

- This offers a basic placeholder when the user enters a request that is not handled (e.g. /xyz12abc)
- Provide a good construction / green themed 404 error page built by someone on your team or ask for a designer or IDM major to create one for your section.

Team 6

Complete work in this order in the views folder:

- **layout.ejs** - Root HTML used by all views.
- header.ejs - Common header & navbar. Used by layout.
- footer.ejs - Common footer. Used by layout.
- index.ejs - default page content for layout body: <%- body %>. Use this during development to show links for GET requests.
-

Always pull first.

Make your modifications.

Add any new files and commit locally.

Push to origin.

Team 6 layout.ejs

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1">
 <title>CGE0x</title>
<!-- add links to css & js here see next slide -->

</head>

<body>
 <% include header.ejs %>
   <div class="container">
     <div class="body">
     <%- body %>

     </div>
   </div>
   <% include footer.ejs %>

</body>
</html>
```

- Create the overall layout view.
- Include header, footer
- Add main replacement area with `<%- body %>`
- Express **request handlers** will use request.render() to replace the content here.

```
api.get('/', (req, res) => {
 res.render('puppy/index.ejs')
})
```

Team 6 layout.ejs

```
<!-- add links to css & js here see next slide -->
<link rel="stylesheet" href="../../css/vendor/bootstrap.min.css">
<link rel="stylesheet" href="../../css/vendor/bootstrap-theme.min.css">
<link rel="stylesheet" href="../../css/main.css">
<!-- add links to icons see ref app -->
<link rel="apple-touch-icon" sizes="57x57" href="../../images/apple-icon-57x57.png">
<link rel="apple-touch-icon" sizes="60x60" href="../../images/apple-icon-60x60.png">
<link rel="apple-touch-icon" sizes="72x72" href="../../images/apple-icon-72x72.png">
<link rel="apple-touch-icon" sizes="76x76" href="../../images/apple-icon-76x76.png">
<link rel="apple-touch-icon" sizes="114x114" href="../../images/apple-icon-114x114.png">
<link rel="apple-touch-icon" sizes="120x120" href="../../images/apple-icon-120x120.png">
<link rel="apple-touch-icon" sizes="144x144" href="../../images/apple-icon-144x144.png">
<link rel="apple-touch-icon" sizes="152x152" href="../../images/apple-icon-152x152.png">
<link rel="apple-touch-icon" sizes="180x180" href="../../images/apple-icon-180x180.png">
<link rel="icon" type="image/png" sizes="192x192" href="../../images/android-icon-192x192.png">
<link rel="icon" type="image/png" sizes="32x32" href="../../images/favicon-32x32.png">
<link rel="icon" type="image/png" sizes="96x96" href="../../images/favicon-96x96.png">
<link rel="icon" type="image/png" sizes="16x16" href="../../images/favicon-16x16.png">
<link rel="manifest" href="../../images/manifest.json">
<meta name="msapplication-TileColor" content="#ffffff">
<meta name="msapplication-TileImage" content="../../images/ms-icon-144x144.png">
<meta name="theme-color" content="#ffffff">
<!-- add script sources -->
<script src="../../js/vendor/jquery-3.2.1.min.js"></script>
<script src="../../js/vendor/bootstrap.min.js"></script>
<script src="../../js/main.js"></script>
</head>
```

Links depend on how you organize your /public folder. Use the correct path for your app.

Find a favicon generator to make a unique icon (and all the versions)

Team 6 header.ejs

```
<div class="container">
 <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
   <div class="container">
     <div class="navbar-header">
       <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
data-target="#navbar" aria-expanded="false"
         aria-controls="navbar">
         <span class="sr-only">Toggle navigation</span>
         <span class="icon-bar"></span>
         <span class="icon-bar"></span>
         <span class="icon-bar"></span>
       </button>
       <a class="navbar-brand" href="/">Welcome</a>
     </div>
     <div id="navbar" class="navbar-collapse collapse">
       <ul class="nav navbar-nav">

         <li>
           <a href="/puppy">Puppies</a>
         </li>
         <% include about.ejs %>
       </ul>
     </div>
   </div>
 </nav>
</div>
```

- The navbar is included in the header.
- To keep it simple, the about menu is kept in a separate file.
- What else needs to change?
- Hint: do we care about puppies in our app?

Team 6 footer.ejs

```
</br>
</br>
<hr>
</br>
<footer class="footer">
 <div class="container">
   <p>&copy; 2017</p>
   <p class="text-muted"><a
href="https://bitbucket.org/professorcase/node-express-mvc-ejs-start">Repository</a></p>
 </div>
</footer>
```

- For convenience, add a link to your repo in the footer.

Team 6 index.ejs

```
<div class="container-fluid">
 <div class=row>
   <div class="col col-md-12">
     <h1>Puppies!</h1>
     <br/>
     <h2>Use this area to display information. </h2>
     <br/>
     <hr>
     <h3>During development, test your api here.</h3>
     <br/>
     <br/>
     <h4>GET a view</h4>
     <ol>
       <li>
         <a href="http://localhost:8089/puppy">http://localhost:8089/puppy</a>
       </li>
     <!-- add more links here -->
     </ol>


   </div>
 </div>
</div>
```

- Index.ejs provides the default body content.
- For now, use it to test all GET requests.

Team 6 logger.js

```
const winston = require('winston')
const fs = require('fs')
const path = require('path')

const logDir = 'logs'

if (!fs.existsSync(logDir)) { fs.mkdirSync(logDir)}

const logger = new (winston.Logger)({
 level: 'debug',
 transports: [
   new (winston.transports.Console)({ json: false, timestamp: true }),
   new winston.transports.File({ filename: path.join(logDir, '/debug.log'),
json: false })
 ],
 exceptionHandlers: [
   new (winston.transports.Console)({ json: false, timestamp: true,
humanReadableUnhandledException: true }),
   new winston.transports.File({ filename: path.join(logDir, '/debug.log'),
json: false, humanReadableUnhandledException: true })
 ],
 exitOnError: false
})

module.exports = logger
```

- All apps need logging.
- We'll use the popular Winston logger.
- When ready, ask Team 1 to enable it in app.js.
- Let Team 3 (controllers) and other teams know that logging functionality is available.
- We won't cover logging in depth. Outside resources are recommended.

Team 6 seeder.js

```
// set up a temporary (in memory) database
const Datastore = require('nedb')
const LOG = require('../utils/logger.js')
const puppies = require('../data/puppies.json') // we don't care about puppies

module.exports = (app) => {
 LOG.info('START seeder.')
 const db = new Datastore()
 db.loadDatabase()

 // insert the sample data into our data store
 db.insert(puppies)

 // initialize app.locals (these objects will be available to our controllers)
 app.locals.puppies = db.find(puppies)

 LOG.debug(`${puppies.length} puppies`)
 LOG.info('END Seeder. Sample data read and verified.')
}
```

- Load seed data.
- Don't load **puppies** - load our estimate data. :)
- We will pre-load our users as well (after we talk about security).
- **Request the list of users from your client.**
- We will not allow any other registered

Team 6 views/estimate

Divide the work among members - and ask for extra help - you have a lot!

1. views/estimate/**create**.ejs
2. views/estimate/**delete**.ejs
3. views/estimate/**details**.ejs
4. views/estimate/**edit**.ejs
5. views/estimate/**index**.ejs
6. views/estimate/**partial_edit**.ejs
7. views/estimate/**select**.ejs

- Team 6 creates the estimate CRUD views.
- You'll get help from other teams to implement everything needed for the view.
- Start with placeholders so we can test our routing.

Team 6 estimate

Divide the work among members - and ask for extra help - you have a lot!

1. views/estimate/**create**.ejs
2. views/estimate/**delete**.ejs
3. views/estimate/**details**.ejs
4. views/estimate/**edit**.ejs
5. views/estimate/**index**.ejs
6. views/estimate/**partial_edit**.ejs
7. views/estimate/**select**.ejs

- Team 6 creates the estimate CRUD views.
- You'll get help from other teams to implement everything needed for each view.
- Start with placeholders in each (see example) so we can test our routing.
- After the h2 appears, use the views from the reference app to progress.

```
<br/>
<br/>
<h2>Estimate details</h2>
<br/>
```

Team 1 enhance app.js

```
// additional requires
const expressLayouts = require('express-ejs-layouts')
const favicon = require('serve-favicon')
const bodyParser = require('body-parser')
const session = require('express-session')
const errorHandler = require('errorhandler')
const dotenv = require('dotenv')
const mongoose = require('mongoose')
const expressValidator = require('express-validator')
const expressStatusMonitor = require('express-status-monitor')
const LOG = require('./utils/logger.js')

// Load environment variables from .env file, where API keys and passwords are
configured.
// dotenv.load({ path: '.env.example' })
dotenv.load({ path: '.env' })
LOG.info('Environment variables loaded.')
}

// configure middleware.....................................................
 app.use(favicon(path.join(__dirname, '/public/images/favicon.ico')))
 app.use(expressStatusMonitor())
```

- Add functionality to app.js as other teams get their files added.
- Read from .env
- Use favicons
- Add a monitor package

Team 1 enhance app.js

```
// log calls
app.use((req, res, next) => {
  LOG.debug('%s %s', req.method, req.url)
  next()
})
// specify various resources and apply them to our application
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({ extended: false }))
app.use(expressValidator())
app.use(express.static(path.join(__dirname, 'public'), { maxAge: 31557600000 }))
app.use(expressLayouts)
app.use(errorHandler()) // load error handler

const routes = require('./routes/index.js')
app.use('/', routes)  // load routing
LOG.info('Loaded routing.')

app.use((req, res) => { res.status(404).render('404.ejs') }) // handle page not
found errors

// initialize data ..........................................
require('./utils/seeder.js')(app)  // load seed data
```

- log all requests
- Set up parsing and validation
- Configure paths and express layouts
- Use an error handler
- Set up routing
- After all routing, if a page isn't found, reply with 404
- Load the seed data

To run

- Requirements: Have **nodejs** & **npm** installed, have **nodemon** installed globally
- Once you have an app.js and a package.json, execute:

```
npm install
nodemon app
```

If errors

- Read the message carefully - sometimes, the messages are very helpful.

```
2017-11-14T15:58:10.979Z - debug: START routing
2017-11-14T15:58:11.307Z - debug: END routing
2017-11-14T15:58:11.307Z - info: Loaded routing.
2017-11-14T15:58:11.354Z - error: uncaughtException: Cannot find module '../data/puppies.json' date=
Tue Nov 14 2017 09:58:11 GMT-0600 (Central Standard Time), pid=13100, uid=null, gid=null, cwd=U:\445
63_f17\cge03, execPath=C:\Program Files (x86)\nodejs\node.exe, version=v8.1.2, argv=[C:\Program File
s (x86)\nodejs\node.exe, U:\44563_f17\cge03\app, app.js], rss=54108160, heapTotal=38887424, heapUsed
=24221544, external=18096197, loadavg=[0, 0, 0], uptime=373406.5357734
Error: Cannot find module '../data/puppies.json'
    at Function.Module._resolveFilename (module.js:485:15)
    at Function.Module._load (module.js:437:25)
    at Module.require (module.js:513:17)
    at require (internal/module.js:11:18)
    at Object.<anonymous> (U:\44563_f17\cge03\utils\seeder.js:4:17)
    at Module._compile (module.js:569:30)
    at Object.Module._extensions..js (module.js:580:10)
    at Module.load (module.js:503:32)
    at tryModuleLoad (module.js:466:12)
    at Function.Module._load (module.js:458:3)
    at Module.require (module.js:513:17)
    at require (internal/module.js:11:18)
    at Object.<anonymous> (U:\44563_f17\cge03\app.js:63:1)
    at Module._compile (module.js:569:30)
    at Object.Module._extensions..js (module.js:580:10)
    at Module.load (module.js:503:32)
[nodemon] clean exit - waiting for changes before restart
```

On open: View Source

- Formatting can be a challenge.
- When the app opens, right-click and "View Source".
- Click each css and js link and verify it is finding the files correctly.
- If any are missing, update the file path and name.

```
7  <!-- add links to local vendor css installed locally -->
8  <link rel="stylesheet" href="../../css/bootstrap.min.css">
9  <link rel="stylesheet" href="../../css/bootstrap-theme.min.css">
10 <!-- our app css -->
11 <link rel="stylesheet" href="../../css/main.css">
12 <!-- add links to icons see ref app -->
13 <link rel="apple-touch-icon" sizes="57x57" href="../../images/apple-icon-57x57.png">
14 <link rel="apple-touch-icon" sizes="60x60" href="../../images/apple-icon-60x60.png">
```

Persistence

In-Memory database

- Our seeder utility is setting up an in-memory database to use during development. Content lasts only as long as the app does.
- We used NeDB - a small NoSQL datastore that mimics MongoDB.

In-Memory database

```
// seeder utility
const Datastore = require('nedb')
const puppies = require('../data/puppies.json')

// we work with app.locals, so use Dependency Injection to inject the app
module.exports = (app) => {

 const db = new Datastore()
 db.loadDatabase()

 // insert the sample data into our data store
 db.insert(puppies)

 // initialize app.locals (these objects will be available to our controllers)
 app.locals.puppies = db.find(puppies)

}
```

File database

- NeDB can also implement file databases.
- Use one datastore / file per collection.
- We will not implement a file database.

```
// Create multiple datastores for multiple collections.
db = {}
db.users = new Datastore('path/to/users.db')
db.robots = new Datastore('path/to/robots.db')

// You need to load each database (here we do it asynchronously)
db.users.loadDatabase()
db.robots.loadDatabase()
```

MongoDB

- Generally, in production, we need better storage.
- We used mongoose for our models to make it easy to move from in-memory storage to MongoDB.
- We will use MongoDB and will help set it up. A complete guide to connecting to MongoDB is not part of this course.

Connecting to MongoDB

```
const mongoose = require('mongoose')  // add require

const isProduction = process.env.NODE_ENV === 'production'

// Connect to MongoDB.........................
if (isProduction) {
 mongoose.Promise = global.Promise
 mongoose.connect(process.env.MONGODB_URI_ATLAS)
 mongoose.connection.on('error', () => {
   console.log('%s MongoDB connection error. Please make sure Atlas MongoDB is running.')
   process.exit()
 })
 LOG.info('Connected to Atlas MongoDB')
} else {
 mongoose.Promise = global.Promise
 mongoose.connect(process.env.MONGODB_URI)
 mongoose.connection.on('error', () => {
   console.log('%s MongoDB connection error. Please make sure local MongoDB is running.')
   process.exit()
 })
 LOG.info('Connected to local MongoDB')
}
```

Configure MongoDB

```
const session = require('express-session')  // add require
const MongoStore = require('connect-mongo')(session) // add require


// configure our datastore.........
app.use(session({
 resave: true,  // save session automatically
 saveUninitialized: true, // create session even if uninitialized
 secret: process.env.SESSION_SECRET, // session cookie is signed with your session secret
 store: new MongoStore({
   url: process.env.MONGODB_URI_ATLAS || process.env.MONGOLAB_URI,
   autoReconnect: true
 })
}))
```

Security

Security

1. Authentication - verifying your identity (who are you?)
2. Authorization - setting rights and permissions to perform certain functions

Passport

- Passport is popular authentication middeware for Node.js
- http://www.passportjs.org/
- We will authenticate using username and password.
- Other methods are available including social authentication through third-party sites such as Facebook and Twitter.
- To use passport, configure:
  1. Authentication strategy
  2. Application middleware
  3. Sessions (optional)

Adding Passport

- Adding passport to an Express app is fairly extensive.
- We have to configure our app, add a new user model, a new user controller, and associated views.
- We will provide a **reference app** that uses passport.
- Use the ref app to implement:
    - config/ensureAuthenticated.js (Team 3: Units 3+10)
    - config/passportConfig.js (Team 3: Units 3+10)
    - controllers/user.js (Team 3: Units 3+10)
    - data/users.json (Team 2: Units 2+11)
    - models/user.js (Team 4: Units 4+9)
    - Seed users in seeder.js (Team 6: Units 6+7)
    - Security in app.js (Team 1: Units 1+12)
    - Team 5: Units 5+8 will add user controller tests

A04

A04

- By the end of this week, each section will have a functional initial version of the app including models & sample data, controllers to handle requests, and initial views (see the **reference app**).
- Initialize your About page to display your name and make sure the AboutController handles a request to your page.
- Our app has calculations and they must update as the client changes data.
- A04 includes views - just not the final versions of the estimate views.
- Your section assistant is your team lead - they have been through this before and are **excellent** resources.
- To earn full credit, we encourage you to attend help sessions and workshops.

Deployment

Heroku

- Web apps aren't Web apps until deployed (to the Web).
- An easy way to get started is Heroku.
- We'll use Heroku to make our project application available to our client.
- The final submission will be graded based on the publicly deployed version.

Heroku Requirements

- Heroku CLI
- free Heroku account
- free Heroku app
- new SSH key (secure shell protocol that links your laptop to the central repo)

Download **PuttyGen** and follow instructions at:
https://www.ssh.com/ssh/putty/windows/puttygen

Heroku

- Go to Heroku's web site.
- Follow instructions to download and install the Heroku CLI (command line interface) 64-bit
- https://devcenter.heroku.com/articles/heroku-cli
- Login: Open command window here as admin & run:
  - > heroku login
  - Provide email and heroku password

Heroku

Adding an alias

- Add a git **heroku** just like **origin**
- Use TortoiseGit / Settings (third option from the end) OR
- $ heroku git:remote -a https://git.heroku.com/cge-f17-staff.git

(use your URL)

Pushing to Heroku

- use TortoiseGit push and switch dropdown OR
- $ git push heroku master