Create an app using some of the express options

$ express movies -e --git

$ cd movies

This already created the following tree structure

├── app.js

├── bin

│   └── www

├── package.json

├── public

│   ├── images

│   ├── javascripts

│   └── stylesheets

│   └── style.css

├── routes

│   ├── index.js

│   └── users.js

├── Summary Of What I did.docx

└── views

├── error.ejs

└── index.ejs

The package.json says "start": "node ./bin/www" for the 1st element, and as value for the “script” key. Here are the content of these files:

|  |  |
| --- | --- |
| **package.json**  {  "name": "02-hw-hobby",  "version": "0.0.0",  "private": true,  "scripts": {  "start": "**node ./bin/www"**  },  "dependencies": {  "cookie-parser": "~1.4.3",  "debug": "~2.6.9",  "ejs": "~2.5.7",  "express": "~4.16.0",  "http-errors": "~1.6.2",  "morgan": "~1.9.0"  }  } | **bin/www**  #!/usr/bin/env node  /\*\*  \* Module dependencies.  \*/  var app = require('../app');  var debug = require('debug')('02-hw-hobby:server');  var http = require('http');  /\*\*  \* Get port from environment and store in Express.  \*/  var port = normalizePort(process.env.PORT || '3000');  app.set('port', port);  /\*\*  \* Create HTTP server.  \*/  var server = http.createServer(app);  /\*\*  \* Listen on provided port, on all network interfaces.  \*/  server.listen(port);  server.on('error', onError);  server.on('listening', onListening);  /\*\*  \* Normalize a port into a number, string, or false.  \*/  function normalizePort(val) {  var port = parseInt(val, 10);  if (isNaN(port)) {  // named pipe  return val;  }  if (port >= 0) {  // port number  return port;  }  return false;  }  /\*\*  \* Event listener for HTTP server "error" event.  \*/  function onError(error) {  if (error.syscall !== 'listen') {  throw error;  }  var bind = typeof port === 'string'  ? 'Pipe ' + port  : 'Port ' + port;  // handle specific listen errors with friendly messages  switch (error.code) {  case 'EACCES':  console.error(bind + ' requires elevated privileges');  process.exit(1);  break;  case 'EADDRINUSE':  console.error(bind + ' is already in use');  process.exit(1);  break;  default:  throw error;  }  }  /\*\*  \* Event listener for HTTP server "listening" event.  \*/  function onListening() {  var addr = server.address();  var bind = typeof addr === 'string'  ? 'pipe ' + addr  : 'port ' + addr.port;  debug('Listening on ' + bind);  } |
| app.js  var createError = require('http-errors');  var express = require('express');  var path = require('path');  var cookieParser = require('cookie-parser');  var logger = x`);  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', 'ejs');  app.use(logger('dev'));  app.use(express.json());  app.use(express.urlencoded({ extended: false }));  app.use(cookieParser());  app.use(express.static(path.join(\_\_dirname, 'public')));  app.use('/', indexRouter);  app.use('/users', usersRouter);  // catch 404 and forward to error handler  app.use(function(req, res, next) {  next(createError(404));  });  // 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; |
| **/routs/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; |
| **/routs/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; |
| **/views/index.ejs**  <!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> |
| **/views/errors.ejs**  <h1><%= message %></h1>  <h2><%= error.status %></h2>  <pre><%= error.stack %></pre> |

So as incomplete as this is I tried to run the server, so I ran the command in my project folder.

$ nodemon

But it gave an error in:

internal/modules/cjs/loader.js:573

throw err;

^

$ npm install

This added 55 packages and created the package-lock.json, also node\_modules with 53 folders in it!!?? (why 53??)

Next, install nodemon so that we don't have to restart our server whenever a change is made

$ npm install nodemon -g

$ nodemon

So now this time nodemon worked with no error, so npm install was actually installing the packages for which we have a require(packageName) statement! And they don’t get installed by just ‘expres’’ command but need ‘npm install’!

In your browser visit [http://localhost:3000 (Links to an external site.)](http://localhost:3000/) and going to this site indeed gives us a screen that says “Express welcome to screen”

$ ls -a

app.js  / bin / node\_modules / package.json / public / routes  / views / .gitignore

Let's look at these one at a time and modify them as we go.

First app.js we will look at this one section at a time.

var express = require('express');  
var path = require('path');  
var favicon = require('serve-favicon'); -🡪 Didn’t have this one???  
var logger = require('morgan');  
var cookieParser = require('cookie-parser');  
var bodyParser = require('body-parser'); 🡪 Didn’t have this one???

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

var users = require('./routes/hobbies);  
//I had this one below too which this tutorial didn’t have:???

var createError = require('http-errors');

var app = express();

There are 4 modules that are new here

serve-favicon

  - a module for displaying the favicon.ico for your site in the browser tab

morgan

  - a request logger middleware with a lot of options

cookie-parser

  - cookie parsing with signatures.  This parses the browser's cookies and populates req.cookies

body-parser

  - body parsing middleware

I put a debugger in the app.js and reran the ‘nodemon’ and indeed it stops s.where and put a console.log(..) after it, but I see that msg is printed but never go to debugger.???

// view engine setup  
app.set('views', path.join(\_\_dirname, 'views'));  
app.set('view engine', 'ejs');

This middleware will set the views path and the engine to 'ejs'

app.use(logger('dev'));  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({ extended: false }));  
app.use(cookieParser());  
app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/', routes);  
app.use('/users', users);

Set the app to use middleware, set the static path for images/css/fonts

Set the routes for now let's remove the app.use('/users', users) line.

// catch 404 and forward to error handler  
app.use(function(req, res, next) {  
 var err = new Error('Not Found');  
 err.status = 404;  
 next(err);  
});

// error handlers

// development error handler  
// will print stacktrace  
if (app.get('env') === 'development') {  
 app.use(function(err, req, res, next) {  
 res.status(err.status || 500);  
 res.render('error', {  
 message: err.message,  
 error: err  
 });  
 });  
}

// production error handler  
// no stacktraces leaked to user  
app.use(function(err, req, res, next) {  
 res.status(err.status || 500);  
 res.render('error', {  
 message: err.message,  
 error: {}  
 });  
});

This is our error handling logic that will render error pages

module.exports = app; ??? is this referring to the file ‘app.js’ OR the local variable app.(cuz it says below “Export this FILE” its confusing. So I think it is indeed the local variable. Cuz in here we have “var indexRouter = require(‘./routes/index” and then in that index.js file we have a local variable called router and at the end of the file we say “module.exports = router” so the RHS argument is not the file or containing folder name, but rather that variable.

So can we have multiple module.exports statements to export multiple variables???

Export the file so that it can be used throughout the app.

bin/www is the configuration for the app and the environment

public contains images / stylesheets / javascirpts

The only file pre-populated here is style.css.  We can remove this file because we will be precompiling with less

In routes, we will remove the users.js file.

Now let's look at 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;

This is a typical routes file that allows us to render something when the route is requested through the browser.  We will leave this as is except change the title to 'Movies'

views/error.ejs

<h1><%= message %></h1>  
<h2><%= error.status %></h2>  
<pre><%= error.stack %></pre>

The error view will give us a message the status and if we are in development mode a stack trace so that we can better debug the error.

views/index.ejs

<!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>

This is a pretty basic page that simply includes our styelesheets/style.css and displays the title

Now that we are familiar with the app we can start building out some CRUD actions.

 First, lets get materialize into the project download the materialize file [here (Links to an external site.)](http://materializecss.com/getting-started.html) and place the .min.css file in in public/stylesheets/

Now in public/stylesheets/style.css add the following to the top of the file

@import 'materialize.min.css';

Now let's make a few partials for the app that can be included throughout all of our pages.  Make a new directory in views called partials.

views/partials/head.ejs

<title>Movies</title>  
<link rel="stylesheet" href="/stylesheets/style.css" />

views/partials/header.ejs

<nav>  
  <div class="nav-wrapper">  
    <a href="/" class="brand-logo">Movies</a>  
    <ul id="nav-mobile" class="right hide-on-med-and-down">  
      <li><a href="/movies">Movies</a></li>  
      <li><a href="/about">About</a></li>  
    </ul>  
  </div>  
</nav>

And finally update views/index.ejs

<!DOCTYPE html>  
<html>  
  <head>  
    <% include ./partials/head %>  
  </head>  
  <body>  
    <header>  
      <% include ./partials/header %>  
    </header>  
    <h1><%= title %></h1>  
    <p>Welcome to <%= title %></p>  
  </body>  
</html>

Now we can easily inject the head and header into all of our pages.  
NOTE how in the navbar has links! i.e. <a href="THE LINK" > </a> elements!!! and so they're relative to what?

so how does the browser know it is relative to the home localhost:3000/hobbies ??? cuz all we have here is /movies appendd to

s.th but what's that s.th?

CRUD actions :

Add the movies routes to app.js

var movies = require('./routes/movies');  
  
....

app.use('/movies', movies);

views/movies.ejs

<!DOCTYPE html>  
<html>  
 <head>  
 <% include ./partials/head %>  
 </head>  
 <body>  
 <header>  
 <% include ./partials/header %>  
 </header>  
 <h1 class='center'>My Movies</h1>  
 <hr />  
 <div class='row'>  
 <div class='col s12 m4'>  
 <h3>Add Movie</h3>

<!-- Note that yet the button below is NOT linked to anything. Also what the heck the method=’post’ is doing below???-->  
 <form action='/movies' method='post'>  
 <input type='text' name='title' placeholder='Title' />

<button class='btn btn-waves' type='submit'>Add</button>  
 </form>  
 </div>  
 <table class='table col m8 s12'>  
 <thead>  
 <th>Title</th>  
 <th>x</th>  
 </thead>  
 <tbody>  
 <% movies.forEach( function(movie) { %>  
 <tr>  
 <td><%= movie.title %></td>  
 <% }); %>  
 </tbody>  
 </table>  
 </body>  
</html>

 We have set our view up to have a form that posts to /movies

A table that lists all the movies

now routes/movies.js

var express = require('express');  
var router = express.Router();  
  
var movies = [  
 { id: 1, title: 'Oceans 11' },  
 { id: 2, title: 'The Hobbit' },  
 { id: 3, title: 'Who framed Roger Rabbit' }  
]  
  
/\* GET movie listings. \*/  
router.get('/', function(req, res) {  
 res.render('movies', { movies: movies });  
});  
  
module.exports = router;

Now that we can retrieve data from the server and display it let's bring in a database to our application.

Our current data is not relational but could at some point grow to be relational.  Due to the possibility of our data becoming relational we don't want to use a NoSQL database which is often popular in node applications.  We will use PostgreSQL because it is particularly difficult to add into a node application and we will soon be in a rails environment where a lot of this work is just done automagically.

$ npm install -g sequelize-cli  
$ npm install sequelize pg pg-hstore  
$ sequelize init  
$ createdb movies\_dev

After that, checking the directory structure (excluding the ‘node\_modules’ folder) we have:

├── app.js

├── bin

│   └── www

├── config

│   └── config.json

├── migrations

├── models

│   └── index.js

├── package.json

├── package-lock.json

├── public

│   ├── images

│   ├── javascripts

│   └── stylesheets

│   ├── materialize.min.css

│   └── style.css

├── routes

│   ├── hobbies.js

│   └── index.js

├── seeders

├── views

│   ├── error.ejs

│   ├── hobbies.ejs

│   ├── hobbies\_v1.ejs

│   ├── index.ejs

│   └── partials

│   ├── head.ejs

│   └── navbar.ejs

Setup config/config.json with our new database

{  
 "development": {  
 "username": "dave",  
 "password": null,  
 "database": "movies\_dev",  
 "host": "127.0.0.1",  
 "port": 5432,  
 "dialect": "postgres"  
 },  
 "test": {  
 "username": "dave",  
 "password": null,  
 "database": "movies\_test", 🡨------------ removed this too  
 "host": "127.0.0.1",  
 "port": 5432,  
 "dialect": "postgres"  
 }  
}

$ sequelize model:create --name Movie --attributes title:string,description:string

Before running the next command,I had to create the database hobbies\_dev or else the next command would give error that database hobbies\_dev does not exist. So to do that had to enter the psql environment by:

$psql –U postgres –h localhost –p 5432

And once in there type:

CREATE DATABASE hobbies\_dev

And then quit the psql environment by typing \q and enter

??? what if the port number is different? What the hell is port number here? Can I create multile hobbies\_dev databases but using different ports?

$ sequelize db:migrate

Sequelize relies on promises which we haven't covered yet so we will have to make some assumptions and abstract some detail away for now.

routes/movies.js

var express = require('express');  
var router = express.Router();  
var Movie = require('../models').Movie;  
  
/\* GET movie listings. \*/  
router.get('/', function(req, res) {  
 Movie.all()  
 .then( function(movies) {  
 return res.render('movies', { movies: movies });  
 })  
});  
  
module.exports = router;

Next, we need to add Movies to our database:

routes/movies.js

/\* POST add movie listing \*/  
router.post('/', function(req, res) {  
 var title = req.body.title;  
 Movie.create({ title: title })  
 .then( function() {  
 res.redirect('/movies');  
 });  
});

Now to finish out the crud actions we need to deal with edit and delete.  A browser can only make GET and POST requests but we want to use the same URL for both update and delete /movies/:id and route to the correct function based on the verb PUT or DELETE.  To do this we need to add middleware to express.

$ npm install method-override --save

app.js

....  
var methodOverride = require('method-override');  
  
....  
app.use(bodyParser.json());  
app.use(methodOverride('\_method'));  
  
....

Now we can add a delete form to our table.

views/movies.ejs

<% var url = "/movies/" + movie.id + "?\_method=DELETE" %>  
 <tr>  
 <td><%= movie.title %></td>  
 <td>  
 <form method="POST" action=<%= url %>>  
 <button class="btn red">Delete</button>  
 </form>  
 </td>  
</tr>

We need a route for delete:

routes/movies.js

router.delete('/:id', function(req, res) {  
 Movie.findById(req.params.id)  
 .then( function(movie) {  
 movie.destroy()  
 })  
 .then( function() {  
 return res.redirect('/movies');  
 });  
});

Add an edit button to our movies.ejs file.

<thead>  
 <th>Title</th>  
 <th>Edit</th>  
 <th>x</th>  
</thead>  
  
....  
  
<% var url = "/movies/" + movie.id + "?\_method=DELETE" %>  
<% var editUrl = "movies/" + movie.id + "/edit" %>  
<tr>  
 <td><%= movie.title %></td>  
 <td>  
 <a class="btn" href=<%= editUrl %>>Edit</a>  
 </td>  
....

Now we need a route that mathes /movies/:id/edit:

routes/movies.js

router.get('/:id/edit', function(req, res) {  
 Movie.findById(req.params.id)  
 .then( function(movie) {  
 return res.render('edit', { movie: movie });  
 });  
});

Next make a file in views called edit.ejs:

views/edit.ejs

<html>  
 <head>  
 <% include ./partials/head %>  
 </head>  
 <body>  
 <header>  
 <% include ./partials/header %>  
 </header>  
 <div class="container">  
 <% var url = "/movies/" + movie.id + "?\_method=PUT" %>  
 <form method="POST" action=<%= url %> >  
 <input required name="title" value=<%= movie.title %> />  
 <a href="/movies" class="btn red">Cancel</a>  
 <button class="btn">Save</button>  
 </form>  
 </div>  
 </body>  
</html>

Finally, we need somewhere to submit this form to the route will be /movies/:id but the method will be a PUT request.

routes/movies.js

router.put('/:id', function(req, res) {  
 Movie.update(  
 { title: req.body.title },  
 { where: { id: req.params.id } }  
 )  
 .then( function() {  
 return res.redirect('/movies');  
 })  
});

It's worth noting that the order changed in the UI and this can be very confusing to the end user.  That's because it is ordering by "updatedAt" instead of "createdAt" we can fix this in our index route.

reoutes/movies.js

router.get('/', function(req, res) {  
 Movie.all({  
 order: [  
 ['createdAt', 'ASC']  
 ]  
 })  
 .then( function(movies) {  
 ....