Express

Chapter 21 in Professional Node.js Textbook

Frameworks

- We started the semester writing web servers on top of raw sockets
- We then learned about connect (Chapter 20), which provided modules for:
  - Logging
  - Parsing the query string
  - Parsing the request body
  - Serve static files (css, js files)
  - Parsing cookies
  - Managing Sessions
- And we also saw how ejs can let us create powerful view templates.

Frameworks

- There is still a bit more to a full-fledged web applications...
  - The structure of our filesystem - where do we put our static content and our application logic?
  - Defining the mappings between URLs and actual code (controllers).
  - Mappings between view templates and URLs

Express builds on connect* to provide these features, plus many more.

* the newest version of express does not require connect, it can use other middleware

Setting up Express

To get started, we install Express

```
npm install express
```

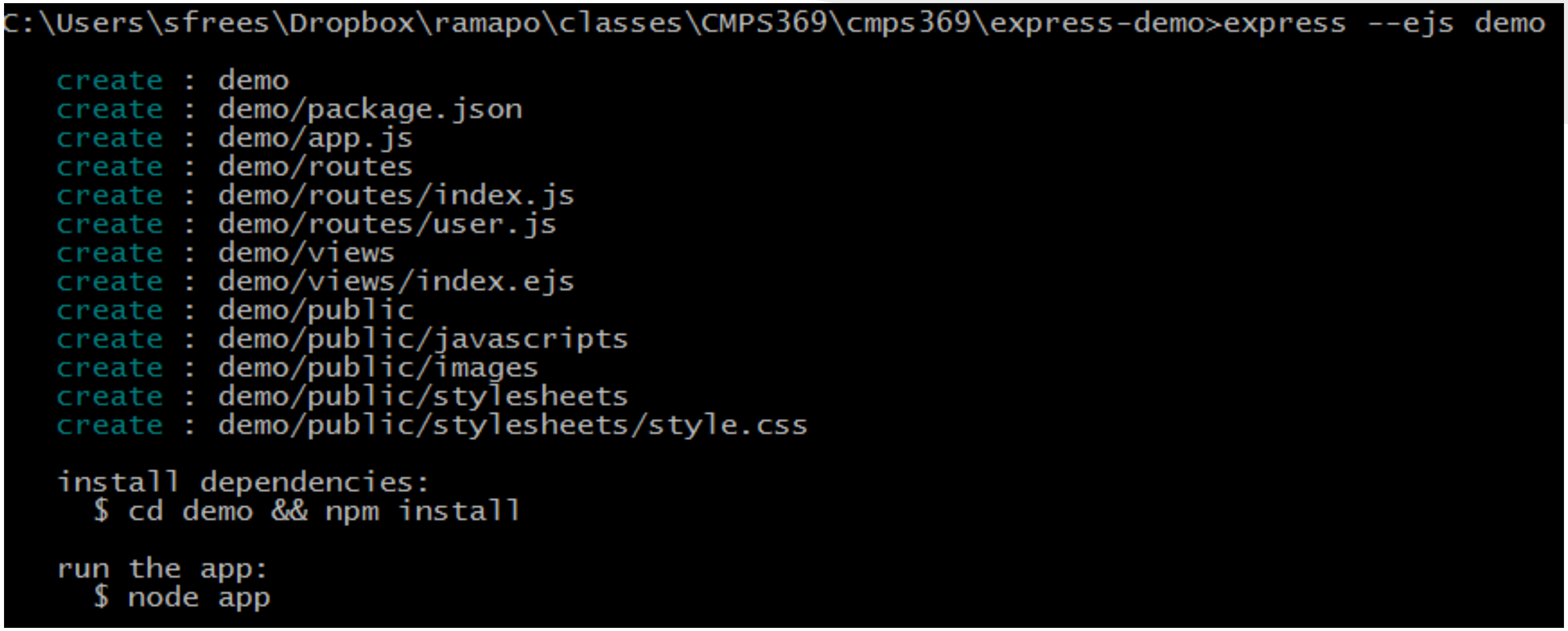
Express can build out the skeleton of a basic app for you, but first you need to install the generator.

```
npm install -g express-generator
```

Next, you can generate an app (called ‘demo’) using the following command:

```
express --ejs demo // the --ejs adds ejs support...
```

Setting up Express

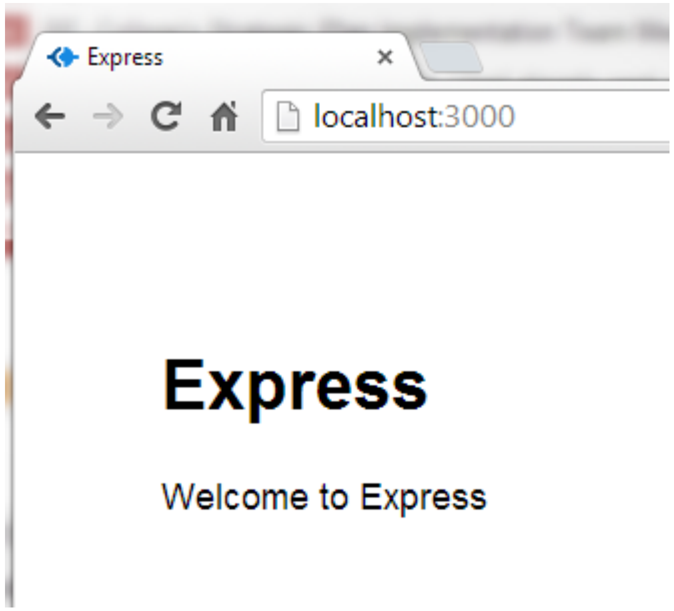


```
:\Users\sfrees\Dropbox\ramapo\classes\CMPS369\cmps369\express-demo>express --ejs demo

   create : demo
   create : demo/package.json
   create : demo/app.js
   create : demo/routes
   create : demo/routes/index.js
   create : demo/routes/user.js
   create : demo/views
   create : demo/views/index.ejs
   create : demo/public
   create : demo/public/javascripts
   create : demo/public/images
   create : demo/public/stylesheets
   create : demo/public/stylesheets/style.css

   install dependencies:
     $ cd demo && npm install

   run the app:
     $ node app
```

Note the directory setup, you just need to do an **`npm install`** to pull the dependencies.

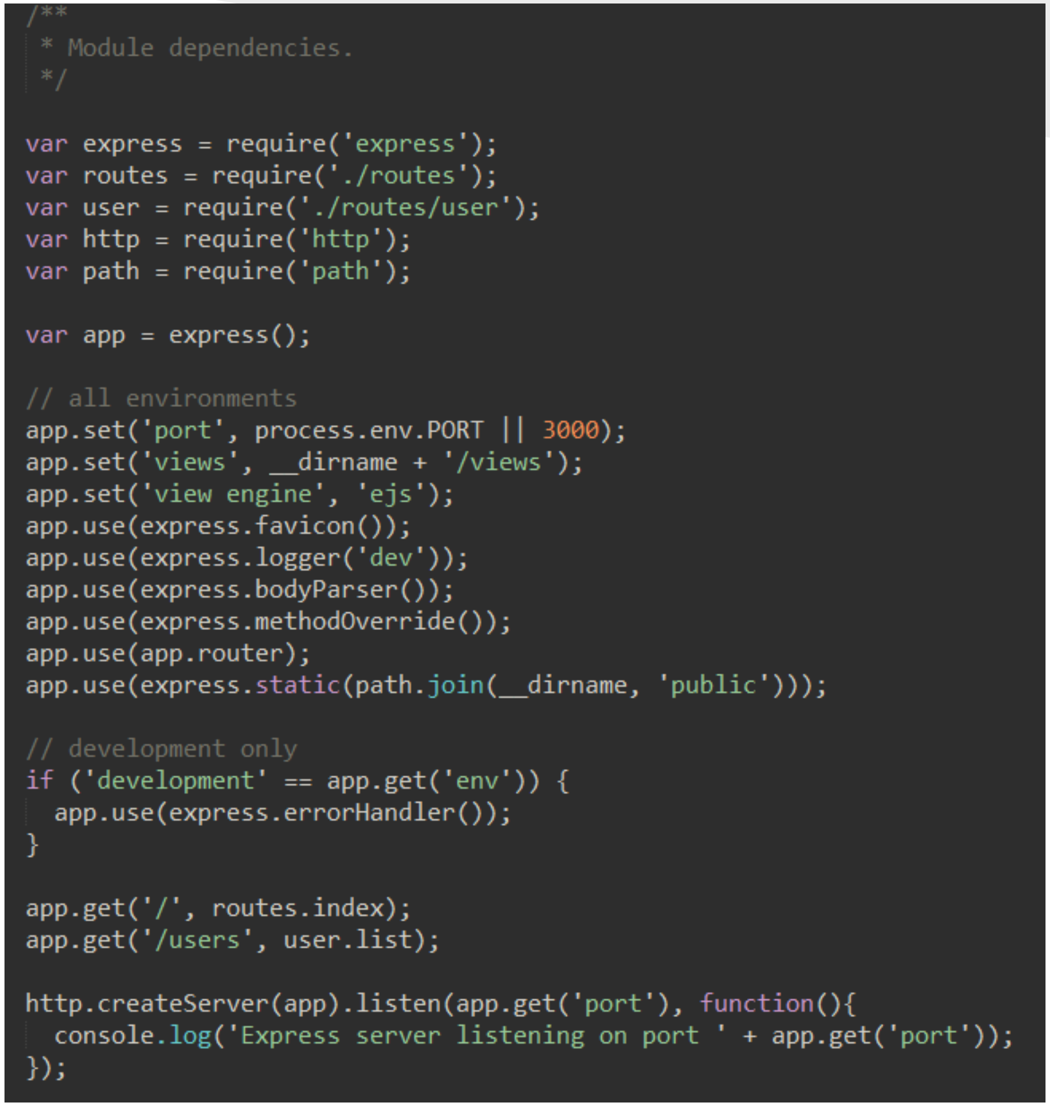
`app.js` contains all the code to start

Default Express app



Express

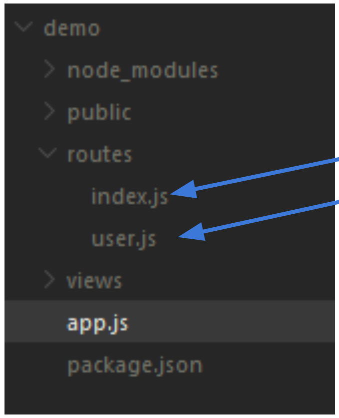
Welcome to Express

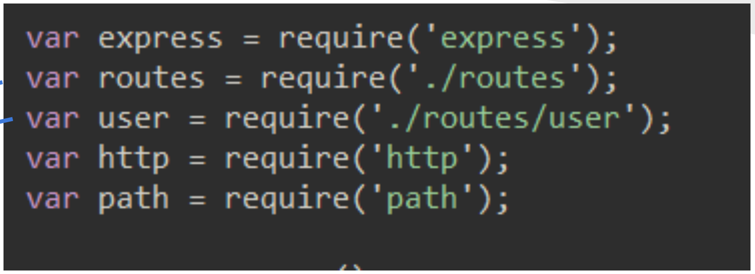


```
/**
 * Module dependencies.
 */

var express = require('express');
var routes = require('./routes');
var user = require('./routes/user');
var http = require('http');
var path = require('path');

var app = express();

// all environments
app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
app.use(express.favicon());
app.use(express.logger('dev'));
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));

// development only
if ('development' == app.get('env')) {
  app.use(express.errorHandler());
}

app.get('/', routes.index);
app.get('/users', user.list);

http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

What just happened?

Directory Structure





```
var express = require('express');
var routes = require('./routes');
var user = require('./routes/user');
var http = require('http');
var path = require('path');
```

We know what require does, but note the route modules

This is actually our “controller” code - we’ll take a look at this shortly.

Note the “.”

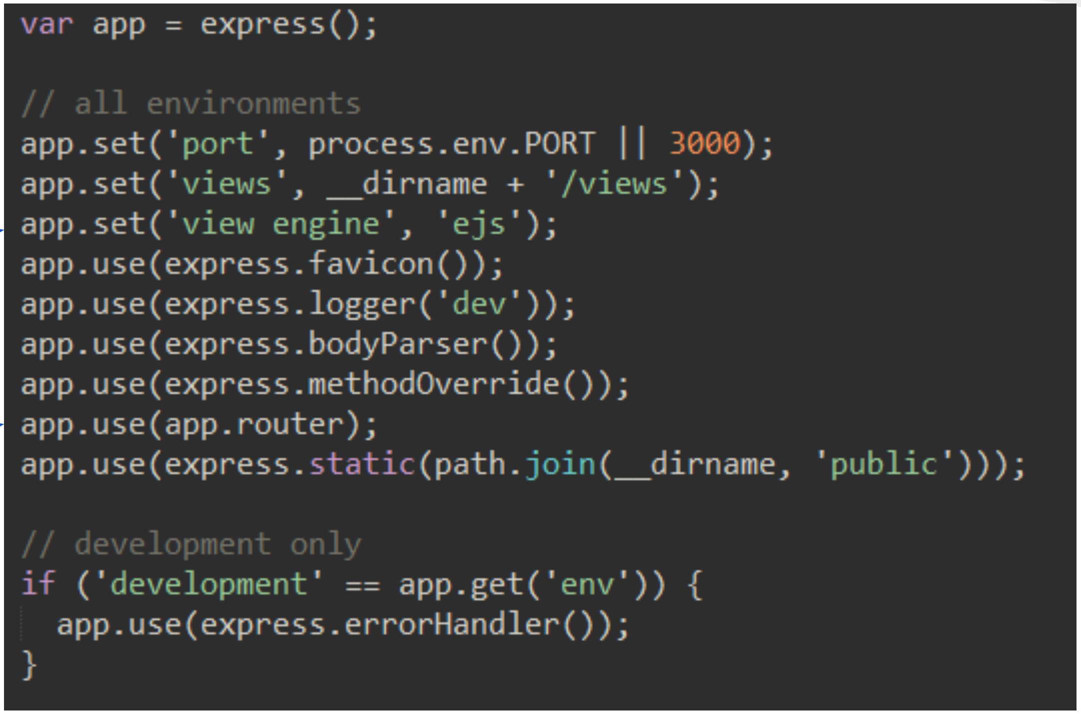
http and path are standard node modules.

What just happened?

This looks a lot like connect, and in fact - some of it is using it!

Will help us render templates

Will tell express to use our router

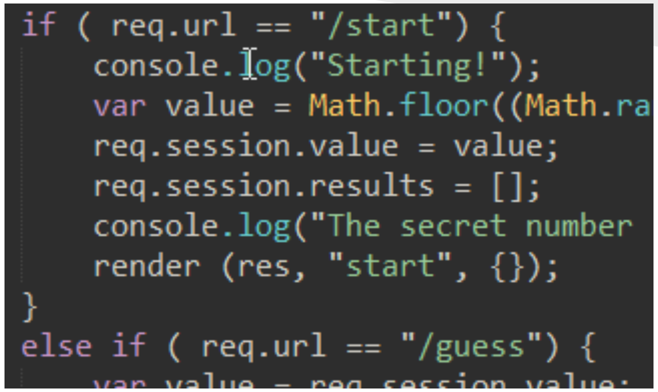


```
var app = express();

// all environments
app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
app.use(express.favicon());
app.use(express.logger('dev'));
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));

// development only
if ('development' == app.get('env')) {
  app.use(express.errorHandler());
}
```

Router?

Recall we typically needed to look at the URL and then branch based on the value.

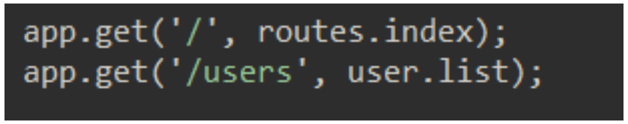
From old guessing game



```
if ( req.url == "/start") {
    console.log("Starting!");
    var value = Math.floor((Math.ra
    req.session.value = value;
    req.session.results = [];
    console.log("The secret number
    render (res, "start", {});
}
else if ( req.url == "/guess") {
    var value = req.session.value;
```

Now we use the express app to declare mappings

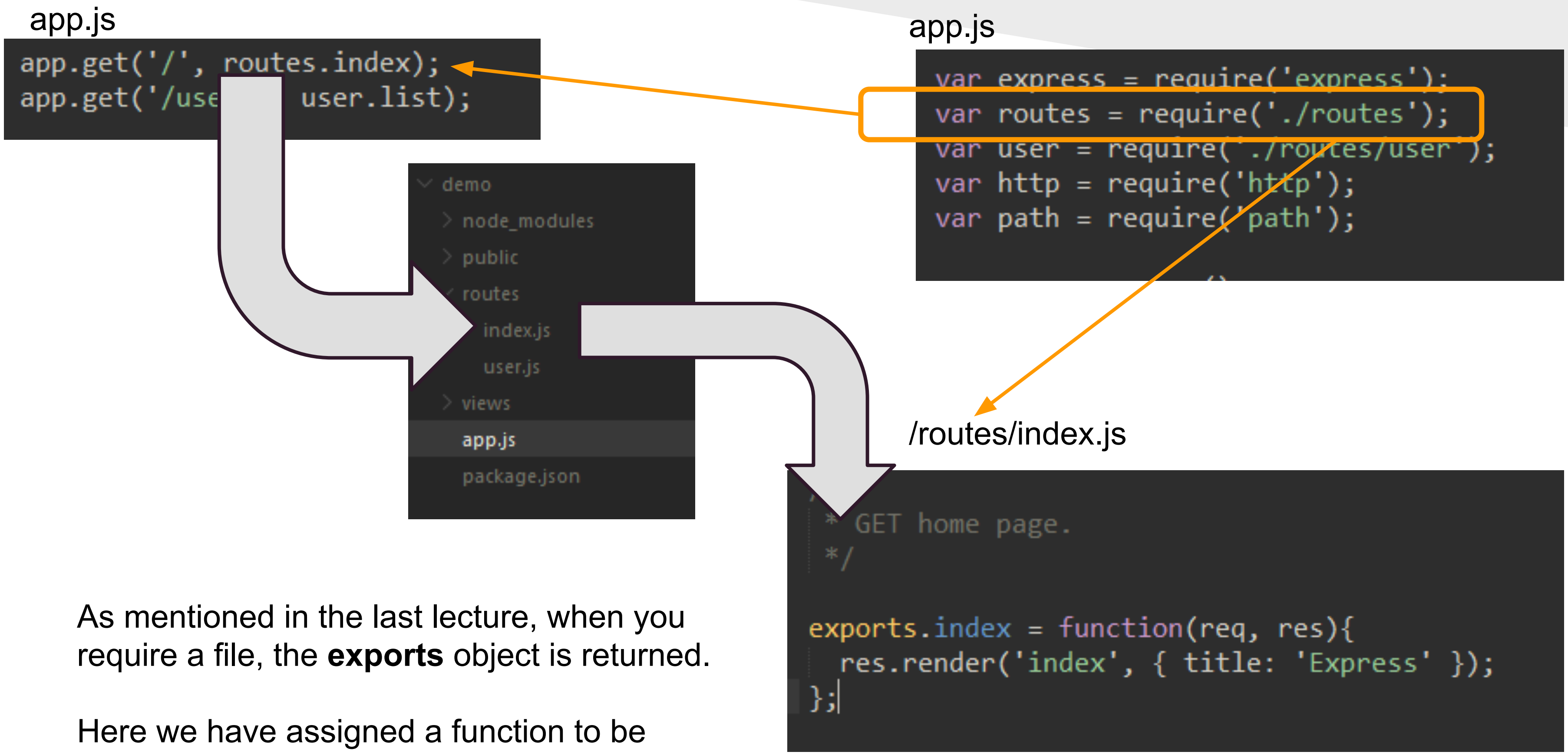
From express's app.js



```
app.get('/', routes.index);
app.get('/users', user.list);
```

With the idea that most apps have uses, express sets up some user management for us. This is helpful in some cases, but we'll get rid of it for now - its not necessary.

Router



```
app.get('/', routes.index);
app.get('/use      user.list);
```

```
var express = require('express');
var routes = require('./routes');
var user = require('./routes/user');
var http = require('http');
var path = require('path');
```

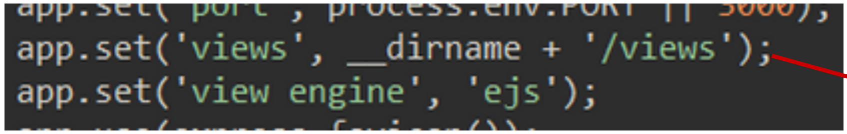
```
demo
  node_modules
  public
  routes
    index.js
    user.js
  views
  app.js
  package.json
```

```
 * GET home page.
 */

exports.index = function(req, res){
  res.render('index', { title: 'Express' });
};
```

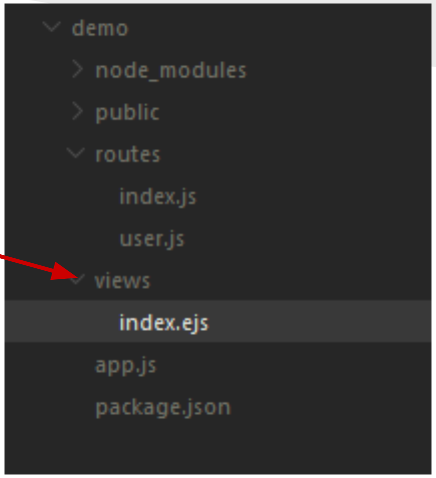
As mentioned in the last lecture, when you require a file, the **exports** object is returned.

Here we have assigned a function to be called whenever a URL request comes in for / (root).

Views



```
app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
```

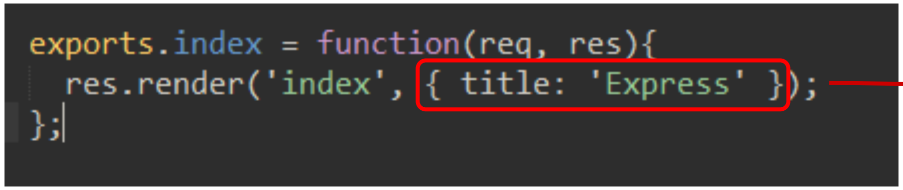


```
demo
  node_modules
  public
  routes
    index.js
    user.js
  views
    index.ejs
  app.js
  package.json
```

When configuring express, we told it our views are in the /views directory

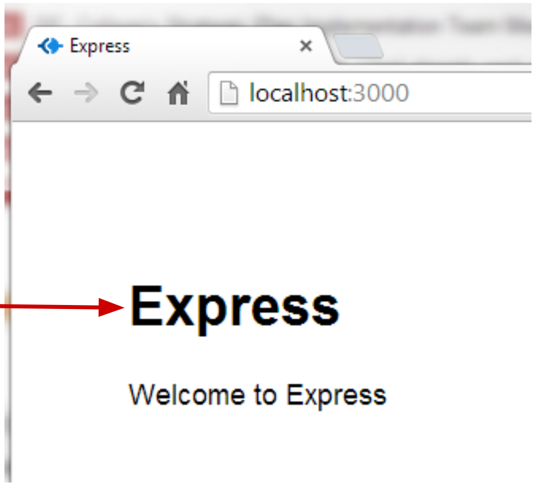
This saves us a little work when rendering - we don't need to explicitly tell express where to find the view, or what extension it is.

It automatically knows we're talking about /views/index.ejs



```
exports.index = function(req, res){
  res.render('index', { title: 'Express' });
};
```

model

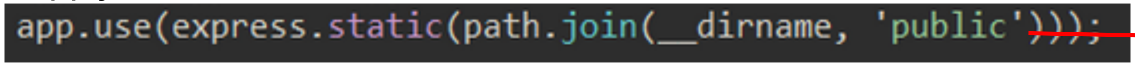


Express

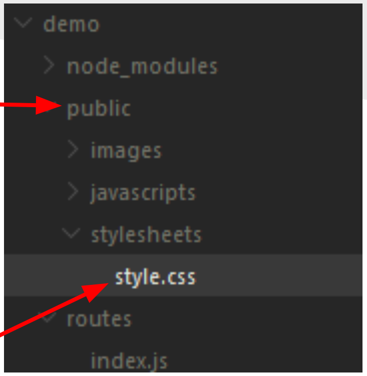
Welcome to Express

Stylesheets

app.js



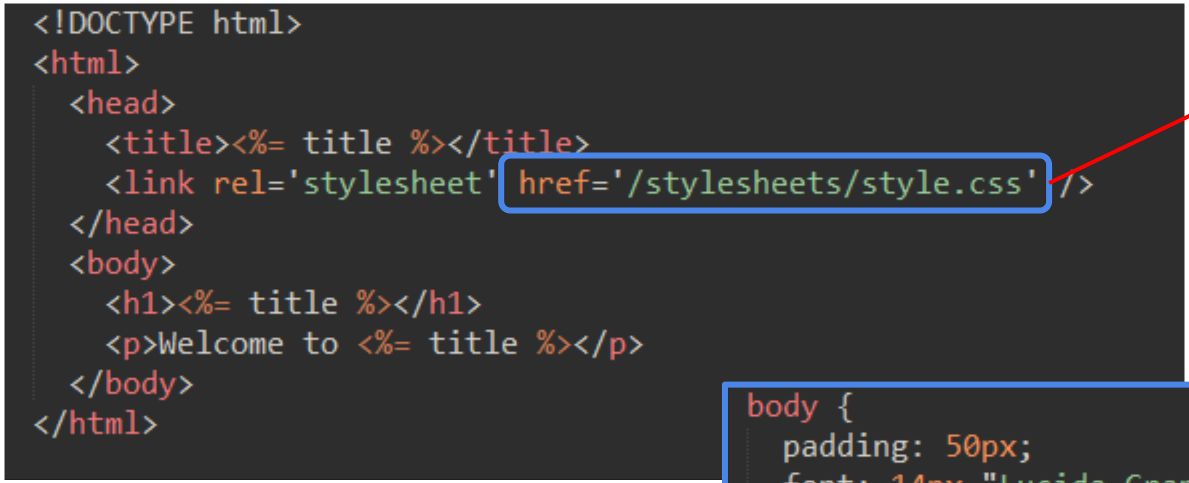
```
app.use(express.static(path.join(__dirname, 'public')));
```



```
demo
  node_modules
  public
    images
    javascripts
    stylesheets
      style.css
  routes
    index.js
```

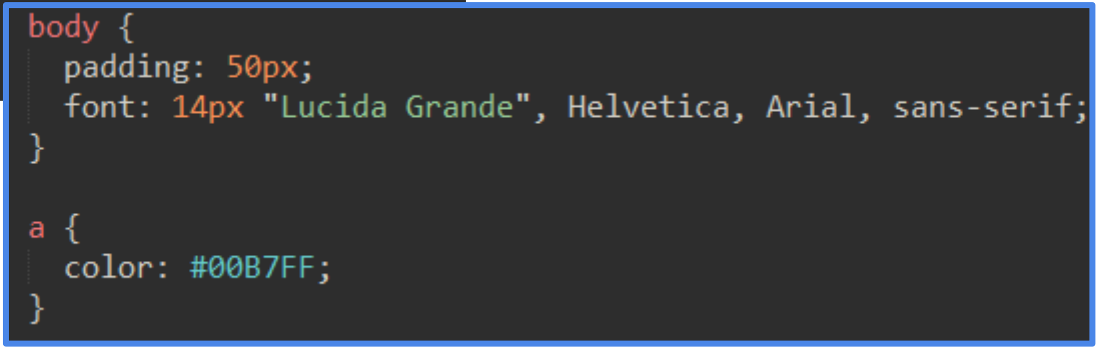
Stylesheets are in the public directory, and are automatically served

index.js



```
<!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>
```

style.css



```
body {
  padding: 50px;
  font: 14px "Lucida Grande", Helvetica, Arial, sans-serif;
}

a {
  color: #00B7FF;
}
```

Guessing Game - Express

Now lets build our familiar guessing game

- We will use sessions
- We will keep track of the guesses
- and we will use express to manage our pages

We'll also add an additional page called "stats" that display the total number of games played (for everyone) and the average number of guesses.

Thats it?

There isn’t much more needed for a basic app - but express’s main advantage is the “extras”

With express, its easy to start using:

- Using alternative template languages - JADE
- Using dynamic style sheets - LESS
  - ... and Bootstrap
- Using authentication - Passport