Cloud Computing (INGI2145) - Lab Session 6

Nicolas Laurent, Marco Canini

Today, we'll continue our exploration of Javascript web programming. In particular, you will be introduced to Express, a simple web application framework that runs on top of Node.js, and which you will use in the second homework assignment. You will also be introduced you to client-side Javascript programming.

Express

To use express, enter the lab06 folder under INGI2145-vm, and have a look at the file called package.json. This file contains a description of your Node.js application including its dependencies. Then install the dependencies with:

```
npm install
```

Now, you can use express locally by writing the following into a Javascript file, that you can run with `nodejs`:

```
var express = require('express');
var app = express();

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

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;
  console.log('Example app listening at http://%s:%s', host, port);
});
```

If you now point your browser to `http://localhost:3000/`, you should see the “Hello World!” message.

What this simple example does is that it defines a “route”. The `app.get` call says that for HTTP `GET` requests on the root address, the server should reply with “Hello, World!”.

What Express does is essentially map the HTTP requests it receives to a callback which will determine the answer to the request. The callback is a Javascript function that receives two parameters: a request object, and a response object. Convenience methods are defined on these objects to help with the usual actions: getting the request’s parameters, sending data as output, etc…

Express’ role is to map each request to multiple callbacks, and to specify for each callback the class of requests it should apply to. Express calls these callbacks “middleware”. The `app.get` call in the examples installs a middleware that applies only to `GET` requests to the root. The more general method to add middleware is `app.use`. It does not allow to discriminate by HTTP verb however, but you could do this filtering inside the middleware itself.

Middleware are executed in the order in which they are specified in the file.

```
app.use(function (req, res, next) {
    console.log('Time:', Date.now());
    next();
});

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

In this example, the same request as before would now invoke the middleware passed to `app.use`, before passing control to the next middleware via the `next()` call. Note that the callback in `app.get` could also take `next` as a third parameter. It is ommitted here only because it is unnecessary (as this is the last middleware in the chain).

Why this emphasis on “middleware”, and why not simply call them “callbacks” or “functions”? The idea is that those callbacks can hide some fairly complex (and potentially stateful) processing. Moreover, since middleware is the unit of processing, they are easy to bundle and share. See this list (http://expressjs.com/resources/middleware.html) to get a sense of what middleware are used for.

Express Exercises

For the exercises, we’ll start with a series of exercises from nodeschool.io (http://nodeschool.io/) , like we did last week.

```
npm install -g expressworks
expressworks
```

These exercises will walk you through a series of simple scenarios using Express and some external middlewares. You will notably learn how to serve static files, render template files and serve them.

If at some point you feel out of depth, don’t hesitate to consult the guides and the

API reference on the official website (http://expressjs.com/4x/api.html) .

EJS

We'll now practice how to use the EJS (https://github.com/tj/ejs) (Embedded Javascript) templating engine. First, read this introduction (http://www.embeddedjs.com/getting_started.html) .

Can you get Express to render a list of supplies like in the example?

Session

We'll now practice how to use the Express session store (https://github.com/expressjs/session) . Add `express-session` to the list of dependencies in package.json and install it with `npm install`.

Try to create a Node.js application that manipulates the session state similar to this example (https://github.com/expressjs/session#reqsession) .

Extra Practice: Node

If you feel you need some more practice with Express, or if you're just interested, you can follow this tutorial (http://code.tutsplus.com/tutorials/introduction-to-express--net-33367) on how to setup a simple blog with Express. It uses handlebars for templating (yes, there are **a lot** (http://garann.github.io/template-chooser/) of Javascript templating engines).

Client-Side Javascript

So far, we've mostly talked about Javascript on the server side. But Javascript is famous mostly for being *the* client-side programming language.

When using Javascript on the client side, what we seek to do is to manipulate the DOM (Document Object Model – in essence, the HTML markup of the page) in response to user actions, without needing to navigate to a new page. Another key feature of client-side Javascript is the ability to download data in the background. This is sometimes called AJAX (Asynchronous Javascript And XML – although nowadays it has little to do with XML). The data obtained this way will often be displayed via DOM manipulation.

There is a standardized API to manipulate the DOM, but it used to be more limited, and to be plagued by browser incompatibilities. For this reason, most people ended up using the jQuery library to manipulate the DOM and handle events.

If you want to learn about jQuery, the documentation (http://learn.jquery.com/using-jquery-core/) is rather good. You can also check the API reference (http://api.jquery.com/) . If you want some hands-on practice, check this codecademy course (http://www.codecademy.com/en/tracks/jquery) . If you want to learn about the vanilla way, read this introduction to the DOM (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction) and this introduction to events (https://developer.mozilla.org/en/docs/Web/API/Event) .

Going up a layer, there are a lot of client-side MVC (Model-View-Controller) and MVVM(Model-View-ViewModel) Javascript frameworks. These help build what is called Single Page Applications (SPA): basically a webpage that acts and feels like a real application - this is made possible by AJAX and DOM manipulation: everything happens on a single page without the need to navigate to other pages.

These apps tend to have a lot more Javascript code than your typical webpage, hence the need for frameworks to organize this code.

If you want to learn about client-side frameworks, a good place to start is TodoMVC (http://todomvc.com/) . This website implements a simple todo-list application in a large number of different frameworks, allowing you to compare the style of different frameworks. It also has a vanilla Javascript implementation, and a jQuery-only implementation.