Tutorial 2

Node.js & Express web framework

>>> Express@1 :: what is web framework?

A **web framework** (**WF**) or **web application framework** (**WAF**) is a software framework that is designed to support the development of web applications including web services, web resources, and web APIs. Web frameworks provide a standard way to build and deploy web applications.

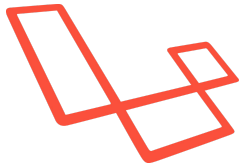
Web frameworks aim to automate the overhead associated with common activities performed in web development. For example, many web frameworks provide libraries for database access, templating frameworks, and session management, and they often promote code reuse. Although they often target development of dynamic web sites, they are also applicable to static websites.

>>> Express@2 :: what is web framework?

Different languages has different web frameworks

Python

C#



PHP

Java

JavaScript

>>> Express@3 :: what is express?

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

recall http core module for creating web servers

>>> Express@4 :: how to run http server using pure node module?

We used ‘http’ node core module to create the server and run it. But it is not so flexible for making complex web applications.

We have to define all of the logic of our application inside a single function. And for growing apps it is suitable in terms of maintainability.

```
var http = require(‘http’)

var server = http.createServer((req,res)=> {

    res.end(‘<h1>Hello World!</h1>’)

})

server.listen(3000)
```

why we need express instead of http?

>>> Express@5 :: why express?

Because http core module does not provide us the required flexibility, we need framework that provides a thin layer of fundamental web application features, without obscuring Node.js features.

The most famous and mostly used web framework for node.js is Express. This framework is built on top http core module and gives easy and flexible programming interface for creating robust web applications.

>>> Express@6 :: basic usage of express

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

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

app.listen(3000)
```

In this example comparing to pure node.js http module we don't have single function where you have to define all of your application logic.

For each request you can define separate handlers in different places. And you do it explicitly for GET and POST methods of the HTTP protocol which makes your code more maintainable and readable.

>>> Express@7 :: how to install express

1. Create project folder
2. Create <package.json> file using <npm init> command inside your project folder
3. Create entry point file inside your project folder. You can name it <app.js>
4. Use <npm install express --save> command to install <express> to your project
5. Require it in your <app.js> using <require()> function. pass ‘express’ name as an argument
6. Create first request handler <app.get(‘/’, function (req, res) { res.send(‘Hello’)})>
7. Start the server on port 3000.
8. Open your browser and go to <127.0.0.1:3000> and see “Hello”

>>> Express@8 :: request and response objects?

Express will create 2 objects (request and response) and pass them to handlers. You could see them in the previous example:

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

request is - coming request from the client (browser)

response is - answer of server to the request coming from client (browser)

>>> Express@9 :: request and response objects' properties

```
req = {
    _startTime      :   Date,
    app             :   function(req,res){},
    body            :   {},
    client          :   Socket,
    complete        :   Boolean,
    connection      :   Socket,
    cookies         :    {},
    files           :    {},
    headers         :   {},
    httpVersion     :   String,
    httpVersionMajor    :    Number,
    httpVersionMinor    :     Number,
    method          :   String,  // e.g. GET POST PUT DELETE
    next            :   function next(err){},
    originalUrl     :   String,     /* e.g. /erer?param1=23¶m2=45 */
    params          :   [],
    query           :   {},
    readable        :   Boolean,
    res             :   ServerResponse,
    route           :   Route,
    signedCookies   :   {},
    socket          :   Socket,
    url             :   String /*e.g. /erer?param1=23¶m2=45 */
}
```

```
res = {
    app             :   function(req, res) {},
    chunkedEncoding:    Boolean,
    connection      :     Socket,
    finished        :   Boolean,
    output          :   [],
    outputEncodings:    [],
    req             :   IncomingMessage,
    sendDate        :   Boolean,
    shouldkeepAlive     : Boolean,
    socket          :     Socket,
    useChunkedEncdoingByDefault    :    Boolean,
    viewCallbacks   :   [],
    writable        :     Boolean
}
```

You can refer to the properties while developing your handler logic. Check next example.

>>> Express@10 :: request and response objects example

app.js

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


app.get('/', function (request, response) {
    response.send('<h1>' + request.query.name.toUpperCase() + ' is happy!</h1>')
})


app.listen(3000)
```

request object has query property and it is and plain javascript object (see prev.slide). And we can populate that object with properties like shown in the picture. We put <?> mark after the url and then data in the format of <key=value>. ex.:

127.0.0.1:3000?name=George

>>> Express@11 :: query property of request object task

1. Create project folder
2. Create <package.json> file using <npm init> command inside your project folder
3. Create entry point file inside your project folder. You can name it <app.js>
4. Use <npm install express --save> command to install <express> to your project
5. Require it in your <app.js> using <require()> function. pass ‘express’ name as an argument
6. Create first request handler <app.get(‘/’, function (req, res) {)>
7. Inside request handler get previously sent query objects number parameter, multiply it by 1000 and send back to user.
8. Start the server on port 3000.
9. Open your browser and go to <127.0.0.1:3000?number=<any_number>. Check result.

>> See you