503106

ADVANCED WEB PROGRAMMING

CHAPTER 2: EXPRESSJS

LESSON 02 – EXPRESSJS

OUTLINE

OUTLINE

RECALL

```
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

ExpressJS - Initial Steps

- Start by creating a new directory
- npm manages project dependencies—as well as metadata about the project—in a file called package.json
- The easiest way to create this file is to run **npm init**
- The first step will be installing Express. Run the following npm command:

```
npm install express
```

ExpressJS - Initial Steps

- Now create the *app.js* file:

```
const express = require('express')
const app = express()
const port = 3000
// custom 404 page
app.use((req, res) => {
   res.type('text/plain')
   res.status(404)
   res.send('404 - Not Found')
})
```

ExpressJS - Initial Steps

```
// custom 500 page
app.use((err, req, res, next) => {
  console.error(err.message)
  res.type('text/plain')
  res.status(500)
  res.send('500 - Server Error')
})
app.listen(port, () => console.log(
'Express started on http://localhost:${port}; ' +
'press Ctrl-C to terminate. '))
```

ExpressJS - Initial Steps

- Let’s add some routes for the home page and an About page.

```
app.get('/', (req, res) => {
   res.type('text/plain')
   res.send('Web Travel');
})
app.get('/about', (req, res) => {
   res.type('text/plain')
   res.send('About Web Travel')
})
```

OUTLINE

ExpressJS – Views and Layouts

- Essentially, a view is what gets delivered to the user. In the case of a website, that usually means HTML.
- A view differs from a static resource (like an image or CSS file) in that a view doesn’t necessarily have to be static.
- Express supports many different view engines that provide different levels of abstraction.
- In this course we will use **Handlebars.**

ExpressJS – Views and Layouts

- To provide Handlebars support, we'll use Eric Ferraiuolo's express-handlebars package:

```
npm install express-handlebars
```

ExpressJS – Views and Layouts

- Then in our *app.js* file, modify the first few lines:

```
const express = require('express')
const expressHandlebars = require('express-handlebars')
const app = express()
// configure Handlebars view engine
app.engine('handlebars', expressHandlebars({
    defaultLayout: 'main',
}))
app.set('view engine', 'handlebars')
```

ExpressJS – Views and Layouts

- Now create a directory called *views* that has a subdirectory called *layouts*.
- Let’s create a layout for our site. Create a file called *views/layouts/main.handlebars*:

```
<!doctype html>
<html>
<head>
<title>Web Travel</title>
</head>
<body>
{{{body}}}
</body>
</html>
```

ExpressJS – Views and Layouts

- Now let’s create view pages for our home page, *views/home.handlebars*:

```
<h1>Welcome to Web Travel</h1>
```

- Then our About page, *views/about.handlebars*:

```
<h1>About Web Travel</h1>
```

- Then our Not Found page, *views/404.handlebars*:

```
<h1>404 - Not Found</h1>
```

- And finally our Server Error page, *views/500.handlebars*:

```
<h1>500 - Server Error</h1>
```

ExpressJS – Views and Layouts

- Now that we have some views set up, we have to replace our old routes with new ones that use these views:

```
app.get('/', (req, res) => res.render('home'))
app.get('/about', (req, res) => res.render('about'))
    // custom 404 page
app.use((req, res) => {
    res.status(404)
    res.render('404')
})
// custom 500 page
app.use((err, req, res, next) => {
    console.error(err.message)
    res.status(500)
    res.render('500')
})
```

OUTLINE

ExpressJS – Static Files and Views

- Express relies on *middleware* to handle static files and views.
- *Middleware* is a concept that will be covered in more detail later.
- The *static* middleware allows you to designate one or more directories as containing static resources that are simply to be delivered to the client without any special handling.
- This is where you would put things such as images, CSS files, and client-side JavaScript files.

ExpressJS – Static Files and Views

- In your project directory, create a subdirectory called *public.*
- Then, in *app.js* before you declare any routes, you'll add the static middleware:

```
app.use(express.static(__dirname + '/public’))
```

- The static middleware has the same effect as creating a route for each static file you want to deliver that renders a file and returns it to the client.
- So let's create an *img* subdirectory inside *public* and put our *logo.png* file in there.

ExpressJS – Static Files and Views

- Now we can simply reference */img/logo.png* (note, we do not specify *public*; that directory is invisible to the client), and the *static* middleware will serve that file.
- Now let’s modify our layout so that our logo appears on every page:

```
<body>
<header>
<img src="/img/logo.png" alt="Web Travel Logo">
</header>
{{{body}}}
</body>
```

- Remember that middleware is processed in order, and static middleware—which is usually declared first or at least very early—will override other routes.

OUTLINE

ExpressJS – Dynamic Content in Views

- The real power of views is that they can contain dynamic information.
- Let’s say that on the About page, we want to deliver a “virtual fortune cookie.”
- In our *app.js* file, we define an array of fortune cookies:

```
const fortunes = [
   "Conquer your fears or they will conquer you.",
   "Rivers need springs.",
   "Do not fear what you don't know.",
   "You will have a pleasant surprise.",
   "Whenever possible, keep it simple.",
]
```

ExpressJS – Dynamic Content in Views

- Modify the view (*/views/about.handlebars*) to display a fortune:

```
<h1>About Web Travel</h1>
{{#if fortune}}
<p>Your fortune for the day:</p>
<blockquote>{{fortune}}</blockquote>
{{/if}}
```

ExpressJS – Dynamic Content in Views

- Now modify the route /about to deliver the random fortune cookie:

```
app.get('/about', (req, res) => {
      const randomFortune =
fortunes[Math.floor(Math.random()*fortunes.length)]
      res.render('about', { fortune: randomFortune })
})
```

Exercises

- Create route to show a table n rows and m columns, each cell contains value of i+j, where i is row, j is column. (Note: n, m is set in code)

Helpers in view

Loop in view

```
<ul class="people_list">
 {{#each people}}
    <li>{{this}}</li>
 {{/each}}
</ul>
```

Example input

```
{ people: [ "Yehuda Katz", "Alan Johnson", "Charles Jolley", ]}
```

else section which will display only when the list is empty.

```
{{#each paragraphs}}
<p>{{this}}</p>
{{else}} <p class="empty">No content</p>
{{/each}}
```

Helpers in view

Access properties with #with

```
{{#with person}}
{{firstname}} {{lastname}}
{{/with}}
```

Example input:

```
{ person: { firstname: "Yehuda", lastname: "Katz", } }
```

Define references

```
{{#with city as | city |}}
    {{#with city.location as | loc |}}
        {{city.name}}  {{loc.north}} {{loc.east}}
    {{/with}}
{{/with}}
```

Exercises

- Create route to show a table n rows and m columns, each cell contains value of i+j, where i is row, j is column. (Note: n, m is set in code; generate HTML in template)

Get data in request

- **Express 4.0 to 4.15**

  ```
  var bodyParser = require('body-parser’)
  app.use( bodyParser.json() ); // to support JSON-encoded bodies
  app.use(bodyParser.urlencoded({ // to support URL-encoded bodies
  extended: true }));
  ```

- **Express 4.16+**

  ```
  app.use(express.json()); // to support JSON-encoded bodies
  app.use(express.urlencoded()); // to support URL-encoded bodies
  ```

OUTLINE

ExpressJS – Examples

- Example 1. Basic usage

```
// basic usage
app.get('/about', (req, res) => {
    res.render('about')
})
```

- Example 2. Response codes other than 200

```
app.get('/error', (req, res) => {
    res.status(500)
    res.render('error')
})
// or on one line...
app.get('/error', (req, res) => res.status(500).render('error'))
```

ExpressJS – Examples

- Example 3. Passing a context to a view

```
app.get('/greeting', (req, res) => {
    res.render('greeting', {
        message: 'Hello esteemed programmer!',
        cookies: req.cookies
    })
})
```

- Example 4. Rendering a view without a layout

```
// the following layout doesn't have a layout file, so
// views/no-layout.handlebars must include all necessary HTML
app.get('/no-layout', (req, res) =>
    res.render('no-layout', { layout: null })
)
```

ExpressJS – Examples

- Example 5. Rendering a view with a custom layout

```
// the layout file views/layouts/custom.handlebars will be used
app.get('/custom-layout', (req, res) =>
    res.render('custom-layout', { layout: 'custom' })
)
```

- Example 6. Rendering plain text output

```
app.get('/text', (req, res) => {
    res.type('text/plain')
    res.send('this is a test')
})
```

ExpressJS – Examples

- Example 7. Adding an error handler

```
// this should appear AFTER all of your routes
// note that even if you don't need the "next" function, it must be
// included for Express to recognize this as an error handler
app.use((err, req, res, next) => {
    console.error('** SERVER ERROR: ' + err.message)
    res.status(500).render('08-error', { message: "you shouldn't have clicked that!" })
})
```

- Example 8. Adding a 404 handler

```
// this should appear AFTER all of your routes
app.use((req, res) =>
    res.status(404).render('404')
)
```

ExpressJS – Examples

- Example 9. Simple GET endpoint returning only JSON

```
const tours = [
    { id: 0, name: 'Hood River', price: 99.99 },
    { id: 1, name: 'Oregon Coast', price: 149.95 },
]
app.get('/api/tours', (req, res) => res.json(tours))
```

ExpressJS – Examples

- Example 10. GET endpoint that returns JSON, XML, or text

```
app.get('/api/tours', (req, res) => {
    const toursXml = '<?xml version="1.0"?><tours>' +
        tours.map(p =>
            `<tour price="${p.price}" id="${p.id}">${p.name}</tour>`
        ).join('') + '</tours>'
    const toursText = tours.map(p =>
        `${p.id}: ${p.name} (${p.price})`
    ).join('\n')
    res.format({
        'application/json': () => res.json(tours),
        'application/xml': () => res.type('application/xml').send(toursXml),
        'text/xml': () => res.type('text/xml').send(toursXml),
        'text/plain': () => res.type('text/plain').send(toursText),
    })
})
```

ExpressJS – Examples

- Example 11. PUT endpoint for updating

```
app.put('/api/tour/:id', (req, res) => {
    const p = tours.find(p => p.id === parseInt(req.params.id))
    if(!p) return res.status(404).json({ error: 'No such tour exists' })
    if(req.body.name) p.name = req.body.name
    if(req.body.price) p.price = req.body.price
    res.json({ success: true })
})
```

ExpressJS – Examples

- Example 12. DELETE endpoint for deleting

```
app.delete('/api/tour/:id', (req, res) => {
    const idx = tours.findIndex(tour => tour.id === parseInt(req.params.id))
    if(idx < 0) return res.json({ error: 'No such tour exists.' })
    tours.splice(idx, 1)
    res.json({ success: true })
})
```

Exercises

- Add API allow search tour and add new tour.