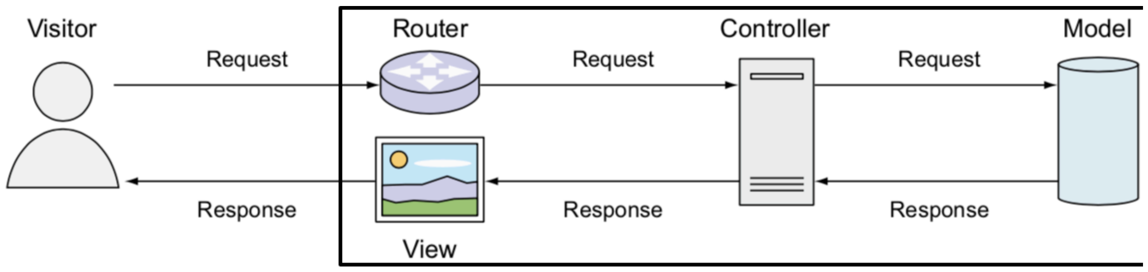
Express

Junghoo Cho

cho@cs.ucla.edu

Express

- A node package for developing a Web server
- HTTP request life-cycle



- Express provides three key functionalities
  - URL-routing mechanism
  - *Middleware* integration
  - *View template engine* integration

Express Demo

```
// -------- app.js ---------
let express = require('express');
let app = express();

app.get('/', (req, res, next) => {
    res.send('Hello World!');
});
app.get('/john', (req, res, next) => {
    res.send('Hello, John!');
});
app.listen(3000);
```

```
$ mkdir demo; cd demo                 // create demo directory
$ npm init -y; npm install express // install express
$ node app.js                         // run app.js
```

URL Routing

- `app.method(path, handler)`

  ```
  app.get('/john', (req, res, next) => { res.send('Hello, John!'
  ```

  - Invoke `handler` for a request sent to `path` (exact match, not prefix) v
  - `app.all(path, handler)` to handle *all* methods
- “Parameters” can be used in the URL path

  ```
  app.get('/dogs/:breed', (req, res, next) => { res.send(req.par
  ```

  - `:breed` makes the matching substring available as a “parameter”
  - Available at `req.params` like `req.params.breed`
- Regular expression may also be used in `path`
- Exact syntax of `path` at https://www.npmjs.com/package/pa

Request Handler

- Takes three parameters: `request`, `response`, `next`

  1. `request`: information on the HTTP request
     - `req.app`: express app that received the request
     - `req.body`: request body
     - `req.query`: (URL) query name value pairs
     - …
  2. `response`: response to be sent to the client
     - `res.send("hello, there!")`
     - More on this later
  3. `next`: the next handler to be called on the request in the *request har*

Request Handling Chain

- Multiple handlers may be attached at the same path
  - When multiple handlers match a request, they are processed in the they are attached
  - *Request handling chain*
- Inside a handler, calling the third parameter `next()` exits fro current handler and moves on to the next in the chain
- If `next()` is not called, the request processing stops there, ig rest in the chain

Generating Response

- Response can be generated using the second parameter `res` the handler
- Status code: `res.status(200)`
- Header: `res.append(field, value)`
- Redirect: `res.redirect([status,] URL)`
  - Send redirect response (default: status 302)
  - Example: `res.redirect('/')`
- Body
  - Can be produced in four different ways. More in the next slide

Generating Body (1)

1. Raw string: `res.send(body)`
   - Send the string `body` as the response
   - Example: `res.send("Hello, world!")`
2. Static file: `res.sendFile(absolute_path)`
   - Send a static file from local filesystem
   - Example: `res.sendFile('/User/cho/public/index.html')`
3. JSON: `res.json(object)`
   - Send JavaScript object `object` in JSON
   - Example: `res.json({title: "Hello", body: "_Love_"})`

Generating Body (2)

4. Generate from Template: `res.render(template-file, tem` `data)`
   - Generate an HTML page from `template-file` using `template-data`
   - Example: `res.render("index", {title: "Hello"})`
   - Multiple *template engines* exist
     - Pug, EJS, Mustache, …
   - We learn EJS (Embedded JavaScript) template engine as an exampl

EJS Template Engine

- A popular template engine used with Express

```
$ npm install ejs
```

- Standard HTML + embedded JavaScript
  - *Scriptlet tag*: similar to JSP
  - `<% ... %>`: javascript for control-flow. no output
  - `<%= ... %>`: print out the result of expression (after HTML escaping
  - `<%- ... %>`: print out the result (raw output. No HTML escaping)

EJS Example (1)

```
<!-- index.ejs --->
<!DOCTYPE html>
<html>
<head><title><%= title %></title></head>
<body>
<ul>
<% for (let post of posts) { %>
    <li><%= post.title %></li>
<% } %>
</ul>
</body>
```

EJS Example (2)

```
// ----  app.js ------
let express = require('express');
let app = express();
app.set('view engine', 'ejs'); // template engine to use
app.set('views', '.')          // view template directory

app.get('/', (req, res, next) => {
    res.render("index",
        { title: "Hello",
          posts: [{title: "Title 1"}, {title: "Title 2"}]
        }
    );
});
app.listen(3000);
```

Advanced URL Routing (1)

- Site structure

```
/birds
    /sparrow
    /dove
/dogs
    /bulldog
    /shepard
```

- Q: For modularity, can we create and use two handlers deper path prefix, one for birds and one for dogs? How?

Advanced URL Routing (2)

- `app.use([path,] middleware)` for prefix routing
  - `path` is interpreted as a *prefix* not exact match
    - `path` prefix is removed in `req.path` passed to middleware (except ending `/` if e
  - `middleware` is a fancy name for request handler
- Example

```
function birds() { ... }
function dogs() { ... }
let express = require('express');
let app = express();
app.use('/birds', birds);
app.use('/dogs', dogs);
app.listen(3000);
```

Modular Middleware

- Q: Inside the middleware, how can we take different actions subpath?
  - `/birds` and `/dogs` are almost like "mini web sites!"
- `express.Router()` to create a "mini web site"
  - Create one `Router()` per prefix, and "mount" them on the correspor
  - Inside each `Router`, use `router.METHOD(subpath, callback)` to ha subpath
  - `Router` is like a "mini Express app"

express.Router() Example

```
// create a router
let birds = express.Router();
birds.get('/sparrow', (req, res, next) => res.send("Sparrow"));
birds.get('/dove', (req, res, next) => res.send("Dove"));

// mount the router at a prefix
app.use('/birds', birds);
```

- Routers can be mounted inside another Router with `use()` t
  "mini mini (?) web site"

Standard Middleware

- Many middleware exists to provide standard functionalities
- `express.static(absolute_path_to_root_dir)`
  - Middleware for serving static files from the file system
- `body-parser` package
  - Collection of HTTP body parsers
  - `bodyParser.json()`: parser for JSON body
- Many more

Middleware Example

```
let express = require('express');
let path = require('path');
let bodyParser = require('body-parser');
let app = express();

app.use('/json', bodyParser.json());
app.use('/www', express.static(path.join(__dirname, 'public')));

app.listen(3000);
```

Error Handling

- Q: What if an error during request handling?
  - Cannot connect to database, file does not exist, …
- A: Call `next(err)` to get into "error handling mode"
  - Stops request handling chain and invokes "error handler"
- An error handler is invoked whenever `next(err)` is called
  - `next()` (no parameter) moves on to the next request handler
  - `next(err)` (single parameter) moves on to the error handler
- *Error handler*: callback function with four input parameters
  - `callback(err, req, res, next)`
  - `err` is what was passed in the call `next(err)`

Error Handler

- Express provides a default error handler
  - Simply prints out `err` passed to `next(err)`
- We can create and use our own error handler instead

```
app.use((err, req, res, next) => {
    res.status(404);
    res.send(err + ": John not found error!");
});
```

  - Attach custom error handler at the end (behind all other regular mi
- Custom error handler can be attached to a specific path
- Multiple error handlers can be attached
  - They are called in sequence if earlier handlers call `next(err)`

Express Application Generator

- Express application generator can be used to generate a ske for express-based server

```
$ mkdir demo; cd demo
$ express --view=ejs // generate skeleton code with EJS templa
$ npm install        // install dependent modules
$ npm start          // execute "start" script in package.json
```

  - `app.js`: main application
  - `public/`: folder for static files
  - `routes/`: route handling middleware
  - `views/`: view template files

MVC in Express

- Skeleton code generated by express application generator p
  nice modular code structure
- Q: Does it follow Model-View-Controller (MVC) pattern?
  - Q: What corresponds to “view”?
  - Q: What corresponds to “controller”?
  - Q: What corresponds to “model”?
- Express application generator provides controller and view, l
  model
  - Create your own separate “module” for data access and managemer

What We Learned

- Popular node.js package to develop a Web application
- Express provides
  - URL routing; request handling chain
  - Middleware integration
  - Template-engine integration
- Express application generator