Web Development with MEAN Stack

CS 144 Web Application

TA: Jin Wang

02/16/2018

Schedule of the following weeks

- Week 6 Node.js + MongoDB + Express
- Week 7 Information Retrieval + Spatial Index
- Week 8 MapReduce Programming
- Week 9 Security (by Zijun Xue)
- Week 10 Final Review

Overview

- Overview of Project 4
- Express - Node.js framework
- Cookie and session

Project 4: Overview

- Goal: Develop Server-side APIs for the markdown editor
- Tasks:
  - Access Control
  - Implement REST APIs
  - Connect with frontend
- Develop Environment
  - Docker Image
  - Express-generator
  - IDE

Note: This part is subject to change! Please refer to the official description posted on course website.

Project 4: Access Control

- Add preview page
  - Show all posts of a user
  - Show a particular post
  - Navigation: *prev* and *next* buttons
- Add user login page
  - User Input: name and password
  - Validate using database
- Check for authorization

```
app.get('/edit', checkLogin);
app.get('/edit', function(req ,res){
 //......
});
```

```
function checkLogin(req, res, next) {
    if(!req.session.user) {
        req.flash('error', 'Not Login!');
        res.redirect('/login');
    }
    next();
}
```

Project 4: Implement REST APIs

- The same functionality with Project 2
- Server APIs
  - GET, POST, PUT, DELETE
- Example

  https://api.github.com/users/hadley/orgs
- Detailed explanation of REST API

https://stackoverflow.com/questions/671118/what-exactly-is-restful-programming

Project 4: Connect with Frontend

- Rewrite the Blog Service part
- Replace operations to local storage with those to MongoDB
- Support URL navigation

Project 4: Develop Environment

- Express-generator
  - Install: *sudo npm install express-generator –g*
  - Usage: *express [option] [project-name]*
  - Run: *npm start*
- IDE: Webstorm
  - An JavaScript IDE with GUI
    https://www.jetbrains.com/webstorm/
  - Belongs to the family of IntelliJ IDEA
  - Note: make sure your program also works in the Docker image

Overview

- Overview of Project 4
- Express - Node.js framework
- Cookie and session

What’s MEAN Stack?

MEAN Stack is a full-stack JavaScript solution that helps you build fast, robust and maintainable production web applications using MongoDB, Express, AngularJS, and Node.js.

Important Programming Issues

- Modules
- Routing
- Asynchronous programming
- EJS template
- MongoDB connection

Modules: basic

- Each file is treated as a module
  - Predefined: package.json -> dependencies
- Require(): load a module
  - **Avoid circular require**
  - It will return empty object {} instead of the actual modules
  - Example: a->b, b->c, c->a
- Exports
  - The initial value of *module.exports* is an **empty object {}**
  - *exports* pointed to the **reference** of *module.exports*
  - require() returns ***module.exports*** *instead of exports*

Modules: example

- Export a module
  - Refer to existing files in ./node_modules/
  - Tip: can be applied to database connection and configuration files
  - Cannot be done in any callback!

a.js

```
const EventEmitter = require('events');

module.exports = new EventEmitter();

// Do some work, and after some time emit
// the 'ready' event from the module itself.
setTimeout(() => {
module.exports.emit('ready');
}, 1000);
```

main.js

```
const hello = require('./a');
hello.on('ready', () => {
    console.log('module a is ready');
});
```

Routing: basic

- Function: app.method(path, handler)
  - Method: HTTP methods, e.g. get, post, all
  - Path: route at which the request will run

    We can have different functions at the same path.
  - Handler: a callback function that executes when a matching request type is found on the relevant route
- Routers:
  - Goal: define all routes in a single file, export as a module
  - Usage: *express.Router()*

Routing: example

index.js

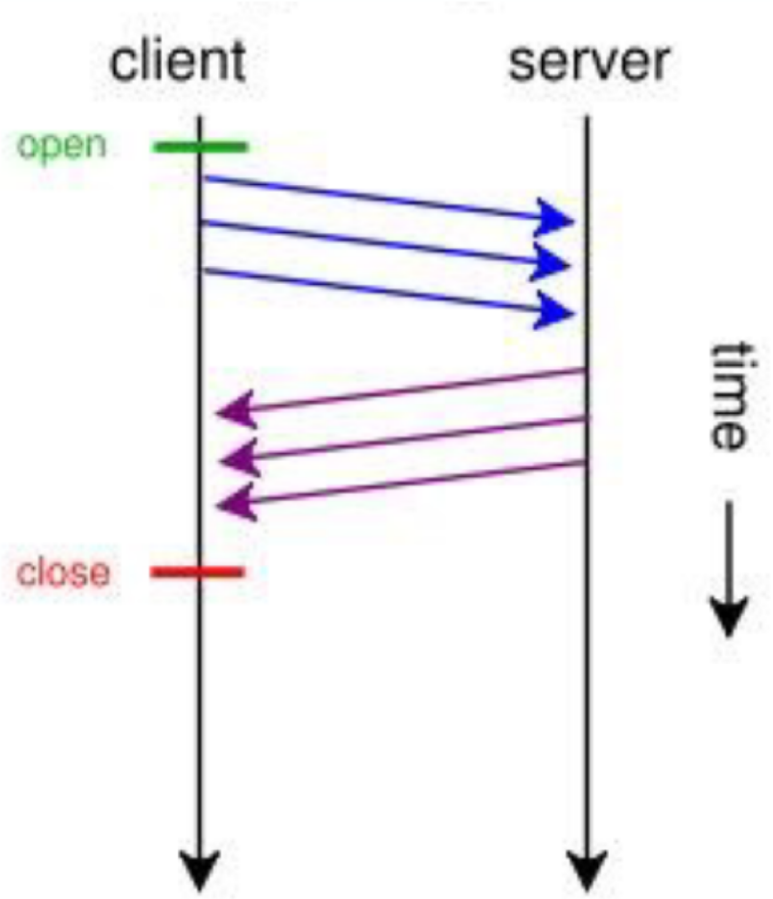
```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res){
    res.send('GET route on things.');
});
router.post('/', function(req, res){
    res.send('POST route on things.');
});

//export this router to use in app.js
module.exports = router;
```

app.js

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

var routes = require('./index.js');

//both index.js and app.js should be in same directory
app.use('/index', routes);

app.listen(3000);
```

Asynchronous Programming

- Calls do not block (or wait) for the call to return from the server.
- Execution continues on in your program, and when the call returns from the server, a "callback" function is executed.
- Application scenarios: database operations, network communication



Asynchronous Programming: Example

- Task: read file from JSON
- Problem: callback hell

```
fs.readFile('config.json',
function (error, text) {
    if (error) {
        console.error('Error while reading config file');
    } else {
        try {
            const obj = JSON.parse(text);
            console.log(JSON.stringify(obj, null, 4));
        } catch (e) {
            console.error('Invalid JSON in file');
        }
    }
});
```

- Solution: Promise

```
import {readFile} from 'fs';

function readFilePromisified(filename) {
return new Promise(
    function (resolve, reject) {
        readFile(filename, { encoding: 'utf8' },
            (error, data) => {
                if (error) {
                    reject(error);
                } else {
                    resolve(data);
                }
            });
    });
}
```

```
readFilePromisified('config.json')
.then(function (text) {
    const obj = JSON.parse(text);
    console.log(JSON.stringify(obj, null, 4));
})
.catch(function (error) {
    console.error('An error occurred', error);
});
```

EJS template

- A template to “translate” JavaScript to HTML

- Setup in app.js

```
app.set('views', path.join(__dirname, 'views'))// set the directory for template
app.set('view engine', 'ejs')// set the template as ejs
```

EJS template

- Template of <% %>
  - <% code %> run the JavaScript without output
  - <%= code %> display the contents after translating to HTML
  - <%- code %> Show the original HTML contents

Example:

Data

```
supplies: ['mop', 'broom', 'duster']
```

Template

```
<ul>
<% for(var i=0; i<supplies.length; i++) {%>
   <li><%= supplies[i] %></li>
<% } %>
</ul>
```

Result

```
<ul>
  <li>mop</li>
  <li>broom</li>
  <li>duster</li>
</ul>
```

MongoDB connection

- User the driver for node.js
  - Start MongoDB service before launching project!
- The mongoose module
  - Connect to database

```
var mongoose = require("mongoose");
mongoose.Promise = global.Promise;mongoose.connect(
    "mongodb://[db-host]:[db-port]/[db-name]");
```

  - Create a database schema and model

```
var nameSchema = new mongoose.Schema({
    firstName: String,lastNameName: String});
var User = mongoose.model("User", nameSchema);
```

  - Save data to database with REST API

```
app.post("/addname", (req, res) => {
 var myData = new User(req.body);
myData.save()
.then(item => {
    res.send("item saved to database");
})
.catch(err => {
    res.status(400).send("save failed");
});
});
```

More details: https://www.npmjs.com/package/mongoose

Overview

- Overview of Project 4
- Express - Node.js framework
- Cookie and Session

Cookie and Session

- Cookie
  - Simple files **sent to client** with request, **stored in client**.
  - HTTP: no status! Cookie: keep the status
  - Track user actions
- Setup cookies

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

app.get('/', function(req, res){
res.cookie('name', 'cookie-name',{maxAge: 360000}).send('cookie set');
});

app.listen(3000);
```

- Delete cookies

```
app.get('/clear_cookie_foo', function(req, res){
    res.clearCookie('foo');
    res.send('cookie foo cleared');
});
```

Session

- Basic idea
  - A “buffer” of data that can be accessed across requests
  - Goal: allow applications to store state
  - Cookie-session: attach cookie to requests: *req.session*
- Module in Express

```
var express = require('express');
var session = require("express-session");
var app = express();
app.use(session({ secret: 'this-is-a-secret-token', cookie: { maxAge: 60000 }}));
app.get('/', function(req, res, next) {
    var sessData = req.session;
    sessData.someAttribute = "foo";
    res.send('Returning with some text');
});
```

Storing Session Data

- In application memory
  - The simplest way
  - The data stored in the lifetime of application runtime.
  - Disadvantages:
    - Failure handling
    - Memory Leaks

Storing Session Data

- In cookies
  - Idea: use cookie to store session data for different users
  - Required modules:
  - Setup:

```
app.use(express.cookieParser('S3CRE7'));
app.use(express.cookieSession());
app.use(app.router);

app.use(express.cookieSession({
    key: 'app.sess',
    secret: 'SUPERsekret'
}));
```

  - Disadvantages
    - Not work well for large, complicated, sensitive data
    - Limited size of cookies a browser can store

Storing Session Data

- In database
  - MongoStore: use MongoDB to store sessions
  - Required Module: connect-mongo

```
var express = require('express');
var MongoStore = require('connect-mongo')(express);

app.use(session({
secret: 'cookie-sec',
key: 'cookie-key',
resave: true,
saveUninitialized: false,
cookie: { maxAge: 1000 * 60 * 60 * 24 },//24 hours
store: new MongoStore({
    db: 'dbname',
    host: 'dbhost',
    port: 'dbportno',
    url: 'dburl'
})
}));
```

Resources

- http://expressjs.com/guide.html
- http://expressjs.com/api.html
- http://github.com/visionmedia/express/tree/master/examples/
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps
- https://www.npmjs.com/package/mongodb
- http://www.embeddedjs.com/