503106

ADVANCED WEB PROGRAMMING

CHAPTER 8: REST APIs and JSON

LESSON 08– REST APIs and JSON

JavaScript Object Notation

JSON: Stands for **J**ava**S**cript **O**bject **N**otation

- Created by Douglas Crockford
- Defines a way of **serializing** JavaScript objects
  - **to serialize**: to turn an object into a string that can be deserialized
  - **to deserialize**: to turn a serialized string into an object

JSON.stringify()

We can use the `JSON.stringify()` function to seralize a JavaScript object:

```
const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing']
};

const serializedBear = JSON.stringify(bear);
console.log(serializedBear);
```

CodePen

JSON.parse()

We can use the `JSON.parse()` function to deseralize a JavaScript object:

```
const bearString = '{"name":"Ice
Bear","hobbies":["knitting","cooking","dancing"]}';

const bear = JSON.parse(bearString);
console.log(bear);
```

CodePen

REST

- REST stands for “representational state transfer,” and the grammatically troubling “RESTful” is used as an adjective to describe a web service that satisfies the principles of REST.
- The formal description of REST is complicated, and steeped in computer science formality, but the basics are that REST is a stateless connection be- tween a client and a server. The formal definition of REST also specifies that the service can be cached and that services can be layered (that is, when you use a REST API, there may be other REST APIs beneath it).

Create API

- We'll plan our API out before we start implementing it. We will want the following functionality:
  - GET /api/attractions
  - GET /api/attraction/:id
  - POST /api/attraction
  - PUT /api/attraction/:id
  - DEL /api/attraction/:id
- Create the model

```
var attractionSchema = mongoose.Schema({
    name: String,
    description: String,
    location: { lat: Number, lng: Number },
    history: {
        event: String,
        notes: String,
        email: String,
        date: Date,
    },
    updateId: String,
    approved: Boolean,
});
var Attraction = mongoose.model('Attraction', attractionSchema);
module.exports = Attraction;
```

Using Express to Provide an API

```
var Attraction = require('./models/attraction.js');

app.get('/api/attractions', function(req, res){
    Attraction.find({ approved: true }, function(err, attractions){
        if(err) return res.send(500, 'Error occurred: database error.');
        res.json(attractions.map(function(a){
            return {
                name: a.name,
                id: a._id,
                description: a.description,
                location: a.location,
            }
        }));
    });
});
```

```
app.get('/api/attraction/:id', function(req,res){
    Attraction.findById(req.params.id, function(err, a){
        if(err) return res.send(500, 'Error occurred: database error.');
        res.json({
            name: a.name,
            id: a._id,
            description: a.description,
            location: a.location,
        });
    });
});
```

```
app.post('/api/attraction', function(req, res){
    var a = new Attraction({
        name: req.body.name,
        description: req.body.description,
        location: { lat: req.body.lat, lng: req.body.lng },
        history: {
            event: 'created',
            email: req.body.email,
            date: new Date(),
        },
        approved: false,
    });
    a.save(function(err, a){
        if(err) return res.send(500, 'Error occurred: database error.');
        res.json({ id: a._id });
    });
});
```

Using a REST Plugin

- Install npm install --save connect-rest
- Import: **var** rest = require('connect-rest');

```
// website routes go here

// define API routes here with rest.VERB....

// API configuration
var apiOptions = {
    context: '/api',
    domain: require('domain').create(),
};

// link API into pipeline
app.use(rest.rester(apiOptions));

// 404 handler goes here
```

```
rest.get('/attractions', function(req, content, cb){
    Attraction.find({ approved: true }, function(err, attractions){
        if(err) return cb({ error: 'Internal error.' });
        cb(null, attractions.map(function(a){
            return {
                name: a.name,
                description: a.description,
                location: a.location,
            };
        }));
    });
});

rest.post('/attraction', function(req, content, cb){
    var a = new Attraction({
        name: req.body.name,
        description: req.body.description,
        location: { lat: req.body.lat, lng: req.body.lng },
        history: {
            event: 'created',
            email: req.body.email,
            date: new Date(),
        },
        approved: false,
    });
    a.save(function(err, a){
        if(err) return cb({ error: 'Unable to add attraction.' });
        cb(null, { id: a._id });
    });
});

rest.get('/attraction/:id', function(req, content, cb){
    Attraction.findById(req.params.id, function(err, a){
        if(err) return cb({ error: 'Unable to retrieve attraction.' });
        cb(null, {
            name: attraction.name,
            description: attraction.description,
            location: attraction.location,
        });
    });
});
```

Test API

- You can test API by using other application like Postman, chrome extension

Cross-Origin Resource Sharing (CORS)

- If you're publishing an API, you'll likely want to make the API available to others. This will result in a *cross-site HTTP request*. Cross-site HTTP requests have been the subject of many attacks and have therefore been restricted by the *same-origin policy*, which restricts where scripts can be loaded from.
- CORS is implemented through the Access-Control-Allow-Origin header. The easiest way to implement it in an Express application is to use the cors package (npm install --save cors). To enable CORS for your application:
- app.use(require('cors')()); or app.use('/api', require('cors')());

Exercise

- Create API for student management
- Create layout for student management and connect with API above