Practical Course: Web Development
REST APIs with NodeJS
Winter Semester 2016/17

Tobias Seitz
Today’s Agenda

• APIs
  – What is it?
  – REST
  – Access Control

• APIs with NodeJS
  – Express
  – StrongLoop / Loopback
  – Adding a datasource

• Hands-On
What is an API

• “Application Programming Interface”
• Interface: Allow other services to use program logic
• Goal: Allow pieces of software to talk to each other
• Characteristics of a Great API:
  – Make it easy for others to use your software.
  – “A Good API needs to appeal to laziness” Kevin Lackner
  – **Intuitive** (make it trivial)
  – **Documented** (if something is not trivial)
  – **Opinionated** (do it the way the API encourages you)
REST API

• Representational State Transfer
• Provide clients access to resources
• Your app manages the states of the resources, but lets other software access the state through the API
• Reasons for using REST APIs [5]:
  – Scalability
  – Generality by using HTTP
  – Independence from other parts of the app
  – Reduced Latency with caching
  – Security with HTTP headers
  – Encapsulation - APIs do not need to expose everything
• Most common format these days: JSON
A Typical API URL

https://www.mywebapp.com/api/v1/things/thing_id
REST API Quick Glance

• Go and look for a REST API

• Examples
  – Spotify
  – Google Maps
  – Flickr
  – Facebook Graph API

• Questions:
  – What do you think makes it a good / bad API?
  – What kind of access control does it have?
  – What kind of restrictions are there?
API Paradigm: CRUD

- **Create**
  ≈ INSERT INTO myData VALUES (....)

- **Read**
  ≈ SELECT * FROM myData WHERE ...

- **Update**
  ≈ UPDATE myData WHERE ...

- **Delete**
  ≈ DELETE FROM myData WHERE ...
REST APIS WITH NODEJS
You should know **Express**

- One of the most popular NodeJS frameworks.
- Characteristics:
  - minimalistic
  - easy to use API
  - many utility methods and middleware functionalities
  - thin layer on top of NodeJS
  - Supports multiple template engines (Pug/Jade, Handlebars, EJS)
- Find the documentation here: [http://expressjs.com/](http://expressjs.com/)
- Package:
  
  `npm install --save express`
- Express generator:
  
  `npm install -g express`
Simple Express App

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

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

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;
  console.log('app listening at http://%s:%s', host, port)
});
```
Express Generator

• Goal: automatically generate the basic structure of an express app that includes views, routes, common dependencies

• Requirements: Install the generator globally:
  $ npm install -g express-generator
  $ express eg-app

• Documentation:
  http://expressjs.com/starter/generator.html

• You still have to install the dependencies manually:
  $ cd eg-app && npm install
Full Stack Solutions

mean.io

The Friendly & Fun Javascript Fullstack for your next web application

MEAN is an opinionated fullstack javascript framework - which simplifies and accelerates web application development.

Get MEAN by running...

```
$ sudo npm install -g mean-cli
$ mean init yourNewApp
```

MEAN stands for:

- **mongoDB**: MongoDB is the leading NoSQL database, empowering businesses to be more agile and scalable.
- **express**: Express is a minimal and flexible node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications.
- **ANGULARJS**: AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.
- **Node.js**: Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.
CRUD with Express

• Example API that manages products.
• Create a new product:
  POST /products
• Retrieve all products:
  GET /products
• Retrieve a particular product:
  GET /product/:id
• Replace a product:
  PUT /product/:id
• Update a product:
  PATCH /product/:id
• Delete a product:
  DELETE /product/:id
Testing POST / PUT / DELETE

- Recommended Tool: Postman [https://www.getpostman.com/](https://www.getpostman.com/)
- Don’t forget the headers, e.g. Content-type: application/json
- Make sure your JSON only uses double quotes
Dummy database: JavaScript Object.

```javascript
var products = {
    'id_A': {
        name: 'Product A',
        price: 30
    },
    'id_B': {
        name: 'Product B',
        price: 50
    }
};
```
GET /products

router.get('/', function(req, res) {
    var productArray = Object.keys(products).map(function(key) {
        var entry = products[key];
        entry.id = key;
        return entry;
    });
    var response = {
        code: 200,
        products: productArray
    };
    res.json(response);
});
Response with all products

Response:
{
   "code": 200,
   "products": [
      {
         "name": "Product A",
         "price": 30,
         "id": "id_A"
      },
      {
         "name": "Product B",
         "price": 50,
         "id": "id_B"
      }
   ]
}

Opinionated:
Products is an Array, instead of an Object literal.
POST /products

```javascript
router.post('/', function(req, res) {
  var entry, id, response;
  if (req.body.name && req.body.price) {
    id = uuid.v1();
    entry = {};
    entry[id] = {
      id: id,
      name: req.body.name,
      price: req.body.price
    };
    products[id] = entry[id];
    response = {
      code: 201,
      message: 'created product',
      products: [entry]
    };
  } else {
    response = {
      code: 1000,
      message: 'missing parameter. required: name, price.'
    }
  }
  res.json(response);
});
```

Intuitive:
Follow API standards
≈ POST creates objects
Response: Product was created

```json
{
    "code": 201,
    "message": "created product",
    "products": [
        {
            "id": "182348e0-abfd-11e6-92a7-4fdc0c2e84f9",
            "name": "Product C",
            "price": 100
        }
    ]
}
```

Intuitive:
Respond with the entire created document, so clients can update their views.
What’s up with this?

• Look at the file /routes/products.js

• Can you think of potential problems for your API?

• How would you solve them?
API Frameworks

- Goal: Simpler, faster creation of APIs and CRUD paradigm for resources
- Often with an abstraction layer
- Popular examples:
  - loopback.io - https://loopback.io/
  - hapi.js - http://hapijs.com/
  - Restify - http://restify.com/
- Comparison: https://strongloop.com/strongblog/compare-express-restify-hapi-loopback/
LoopBack

- Now part of StrongLoop Arc (IBM)
- Installation:
  
  `npm install -g strongloop`
- Getting started wizard:
  
  `slc loopback`
  
  - api-server: already contains authentication methods
  - empty-server: most basic setup
  - hello-world: small working sample
  - notes-app: full working example for a note-taking api
Step 1: Set up the project

```
spengler:04-apis Tobi$ slc loopback

    -----
   |    |
   |---(o)---|
   `--------'
       (  _`U`_ )
   /_____\    /
   |  ~  |
   `-'.___.'__`
      `\  \ `'

What's the name of your application? loopback-api
Enter name of the directory to contain the project: loopback-api
create  loopback-api/
   info  change the working directory to loopback-api

Which version of LoopBack would you like to use? 2.x (stable)
What kind of application do you have in mind? hello-world
Generating .yo-rc.json ...
```
Step 2: Create a model

spengler:loopback-api Tobi$ slc loopback:model
? Enter the model name: product
? Select the data-source to attach product to: db (memory)
? Select model's base class PersistedModel
? Expose product via the REST API? Yes
? Custom plural form (used to build REST URL): products
? Common model or server only? common
Let's add some product properties now.
Step 3: Add properties

Enter an empty property name when done.

Property name: name

invoke loopback:property

Property type: string

Required? Yes

Default value[leave blank for none]:

Let's add another product property.

...
Step 4: Run the app

```bash
spengler:loopback-api Tobi$ node .
```

Web server listening at: http://0.0.0.0:3000
Browse your REST API at http://0.0.0.0:3000/explorer
Supported Methods

localhost:3000/api/products

Use Postman to add some data...

Response:
{
  "name": "Product A",
  "price": 10,
  "id": 1
}
Persisting Models to a Database

• Loopback allows using “connectors” for various databases
• MySQL connector:
  npm install --save loopback-datasource-juggler loopback-connector-mysql
• Getting started:
  slc loopback:datasource
• This is not a trivial step, so you really need to try this yourself.
• Links:
  – https://github.com/strongloop/loopback-connector-mysql
Add a MySQL Datasource

spengler:loopback-api Tobi$ slc loopback:datasource
? Enter the data-source name: mysql
? Select the connector for mysql: MySQL (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings (eg: mysql://user:pass@host/db):
? host: localhost
? port: 3306
? user: pwp
? password: **************
? database: pwp

This will add a new entry to server/datasources.json
server/model-config.json

```json
{
  ...
  "product": {
    "dataSource": "mysql",
    "public": true
  }
}
```
Things to note at this point

• If you try to run the app now, you will get an error.
• **Problem:** There is no table “products” in your database
• **Goal:** You want LoopBack to generate this table for you.
• **Solution:** Automigration.
• Automigration also works, if you want to switch the database (e.g. replace MySQL with Cloudant)
Automigration

```javascript
var path = require('path');
var app = require(path.resolve(__dirname, '../server/server'));

var ds = app.datasources.mysql;
ds.automigrate('product', function(err) {
  if (err) throw err;
  var products = [
    { name: 'Product A', price: 10 },
    { name: 'Product B', price: 50 }
  ];
  products.forEach(function(product, i) {
    app.models.product.create(product, function(err, model) {
      if (err) throw err;
      console.log('Created: ', model);
      if (i === products.length - 1) {
        ds.disconnect();
      }
    });
  });
});
```
Perform Automigration

spengler:loopback-api Tobi$ node bin/automigrate.js

Created:  { name: 'Product A', price: 10, id: 1 }
Created:  { name: 'Product B', price: 50, id: 2 }
After Automigration: We have a table!
API for your project

• Think of a Resource that is going to be accessible through your project API
• Try to model it
  – properties
  – datatypes
• Perform all steps with loopback
Things that we couldn’t cover

- Autodiscovery of Schemas (LoopBack)
- Securing an API
- Manual Deployment and Configuration
- Process Management and Proxies
- Dockerizing a NodeJS app
- .... and much more.

- ==> We’ll get there, when we need them during the project phase.
Personal Experiences

• Put a lot of work into designing and specifying your API. API changes can break much of the applications using the interface.

• You don’t want to maintain a lot of different versions of the API, so it’s better to plan ahead.

• Make sure to bundle API calls on the front end ➔ Only one module contains API information. The module then exports methods to use the API across the entire front end.
Links ‘n’ Stuff

Must read:

Should read:

Wouldn’t do any harm:
2. https://www.youtube.com/watch?v=heh4OeB9A-c
3. https://www.youtube.com/watch?v=qCdpTji8nxo
4. https://www.youtube.com/watch?v=hdSrT4yjS1g
5. https://stormpath.com/blog/fundamentals-rest-api-design
Links ‘n’ Stuff

• http://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/
• http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api
• https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
• https://github.com/RestCheatSheet/api-cheat-sheet#api-design-cheat-sheet