Web Dev II

Back-end Development

By Chase Haddleton

A Little Bit About Me

- Full time student
- Full-stack dev
- Full time coffee addict
- Favourite back-end stack: Node.JS

Why're We Here?

- Designing
- Creating
- Deploying

Our Approach

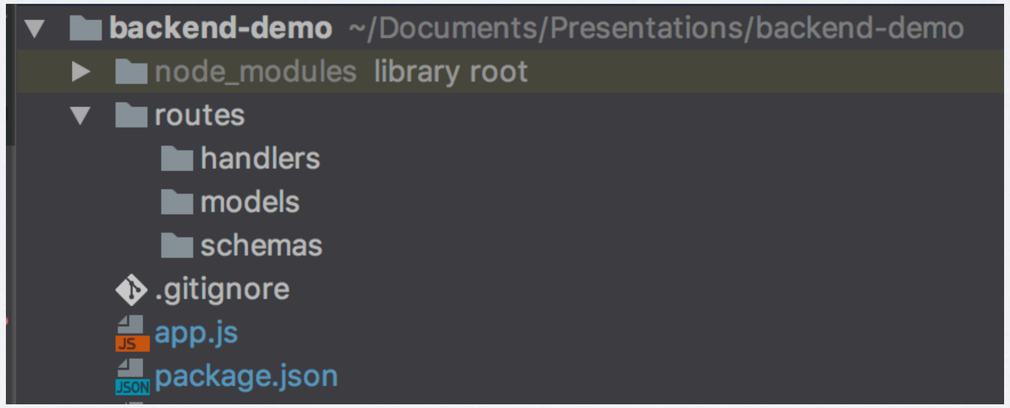
- Challenge-based learning
  - Engage—What are the big concepts?
  - Investigate—I want to do *X*, but I need to know *Y*
  - Act—I know X, I know Y, now I can build

Challenge

- Create & deploy a very simple bank API
  - Create account, create transaction, get account
  - Node.JS, MongoDB, Mongoose as an ORM

Before We Go Further

- Create the following folders and structure



- https://github.com/chasehaddleton/web-dev-ii-htn-2017

Who's Responsible For What?

- Front-end: Interpreting and presenting data
- Back-end: Holding, modifying, and serving data

How Do They Communicate

- REpresentational State Transfer Application Programming Interface (REST API)
- Websocket
- GraphQL API

Where's Our Data

- SQL
  - MySQL, PostgreSQL, Oracle DB, SQL Server, CockroachDB
- NoSQL
  - MongoDB, HBase, BigTable, CouchDB, Redis

How Do You Access Data?

- Database connectors (raw queries)
- Object Relational Mapping
- Object Document Mapping
  - Mongoose for MongoDB

Guiding Principiles

- Uniform
- Stateless
- Client-Server Architecture
- Cacheable
- Layered

RESTful API

GET PUT POST DELETE

What About Hypermedia?

- Interesting in principle
- ‘Self documentation’ is unrealistic
- Clients do not exist

```
{
  "accountNumber": 2342342,
  "balance": {
   "value": 15000.91,
   "currency": "CAD"
  },
  "links": [
   {
     "rel": "self",
     "type": "GET",
     "href": "fakeBank.com/account/2342342"
   },
   {
     "rel": "deposit",
     "type": "PUT",
     "href": "fakeBank.com/account/2342342/deposit",
     "schema": "fakeBank.com/schema/account/deposit"
   }
  ]
}
```

What Do Endpoints Look Like?

- Resources available at descriptive paths
  - Think nouns, not verbs
- HTTP Verbs used on resources (GET, POST, DELETE…)

Design Suggestions

- Define your resources first
- Determine what operations they require
- Design the relationship between the entities require

Best Practices

- Predictability > creativity
- Be consistent & descriptive
- Use HTTP Status codes as indicators

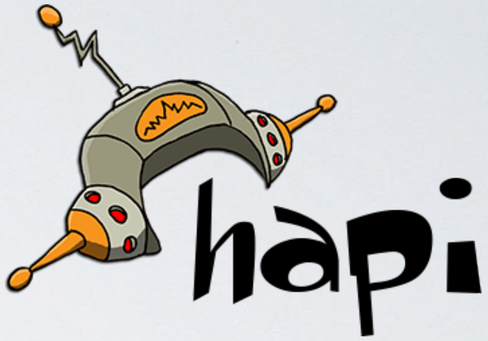
Where Are The Endpoints

- One server
  - Easy to maintain
- Multiple servers
  - Easy to develop with team
  - Servers and language can be chosen for tasks based off responsibilities
  - API gateway to connect them all

Creating A REST API

- Node.js
- Spring Boot
- Flask
- Go

Suggested Node Libraries



- Hapi.JS
  - Provides an extremely easy way to make a REST API
  - Built off Express

```
npm install -save hapi glob path bluebird
```

```
// app.js
const Hapi = require('hapi');
const glob = require('glob');
const path = require('path');
global.Promise = require('bluebird');

const server = new Hapi.Server({
   debug: {
      request: ['error']
   }
});

server.connection({
   port: process.env.PORT || 8080,
   routes: {
      cors: true
   }
});

glob.sync('routes/*.js', {
   root: __dirname
}).forEach(file => {
   const route = require(path.join(__dirname, file));
   if (route.constructor === Array) {
      route.forEach(r => {
         server.route(r);
      });
   } else {
      server.route(route);
   }
});

server.start(err => {
   if (err) {
      console.error(err);
      throw err;
   }
});
```

Import

Initialize the server

Create a connection

Grab every route, add it to the router

Start the server

```
// route/accountRoutes.js

module.exports = [{
  method: 'POST',
  path: '/account',
  handler: (request, reply) => {
    // do something
  }
}];
```

HTTP Verb

Request Path
(case sensitive)

Request handler
request holds request
data, *reply* a call back

Why is this an array?

```
// app.js

glob.sync('routes/*.js', {
  root: __dirname
}).forEach(file => {
  const route = require(path.join(__dirname, file));

  if (route.constructor === Array) {
    route.forEach(r => {
      server.route(r);
    });
  } else {
    server.route(route);
  }
});
```

Checks if the export is an array, iterate & add each route

```javascript
// route/accountRoutes.js

module.exports = [
  {
    method: 'GET',
    path: '/account/{id}',
    handler: (request, reply) => {

    }
  },
  {
    method: 'POST',
    path: '/account',
    handler: (request, reply) => {

    }
  },
  {
    method: 'PUT',
    path: '/account/{id}/transaction',
    handler: (request, reply) => {

    }
  }
];
```

Handlers

- Messy to have them inline in route declarations
- Move to a separate file, import as required

```
// route/accountRoutes.js

const accountHandlers = require('./handlers/accountHandlers');

module.exports = [{
  method: 'POST',
  path: '/account',
  handler: accountHandlers.createAccount
}];
```

Handler Import

Pass as call back

Mongoose And MongoDB

- Schema: definition of data structures
  - Can have validation, custom functions, set and get formatters, etc
- Models: class derived from a defined Schema
  - Methods to find, save, update, etc

```
npm install -save mongoose
```

```
// app.js

const Hapi = require('hapi');
const glob = require('glob');
const path = require('path');
global.Promise = require('bluebird');
const mongoose = require('mongoose');

/**
 * stuff is here
*/

server.start(err => {
  if (err) {
    console.error(err);
    throw err;
  }

  mongoose.connect("mongodb://localhost/bankDemo", {useMongoClient: true}, (err) => {
    if (err) {
      console.error(err);

      throw err;
    } else {
      console.info('MongoDB connected');
    }
  });

  mongoose.Promise = global.Promise;
});
```

Connect to the MongoDB server

Make Mongoose use the standard promise

```
// route/models/Account.js

const mongoose = require('mongoose');
const uuid = require('uuid/v4');

let transactionSchema = new mongoose.Schema({
   amount: Number,
   date: {
      type: Date,
      default: Date.now()
   }
});

let accountSchema = new mongoose.Schema({
   accountId: {
      type: String,
      default: uuid()
   },
   balance: {
      type: Number,
      default: 0,
      set: setBalance
   },
   transactions: [transactionSchema],
   currency: {
      type: String,
      enum: ['CAD', 'USD'],
      default: "CAD"
   }
});

function setBalance(val) {
   return parseFloat(val.toFixed(2));
}

module.exports.model = mongoose.model("Account", accountSchema);
module.exports.schema = accountSchema;
```

Schema creation

Create & export model

Embedding Vs Ref

- Sub-Documents can have schema embedded or be referenced by ObjectID

Sub-Document

- Small document
- Infrequently changed
- Eventual consistency
- Required frequently, fast read

Reference

- Large document
- Frequently changed
- Immediate consistency required
- Frequently not required, slower

```
let transactionSchema = new mongoose.Schema({
  amount: Number,
  date: {
    type: Date,
    default: Date.now()
  }
});
mongoose.model("Transaction", transactionSchema);

let accountSchema = new mongoose.Schema({
  id: {
    type: String,
    default: uuid()
  },
  balance: {
    type: Number,
    default: 0,
    set: setBalance
  },
  transactions: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: "Transaction"
  }],
  currency: {
    type: String,
    required: true,
    enum: ['CAD', 'USD']
  }
});
```

What Do We Have So Far?

- Routes
- Handler (separated from our routes)
- A model

Creating And Savings

- Import
- Instantiate
- Set
- Save

```
// route/handlers/accountHandlers.js

const Account = require('../models/Account').model;

function newAccount(request, reply) {
   let account = new Account({
      currency: “CAD"
   });

   account.save();
   reply({
      successful: true
   });
}

module.exports = {
   newAccount
};
```

Hapi.Js Request Object

- GET data is stored in params
- POST data is stored in payload

```javascript
// route/handlers/accountHandlers.js

const Account = require('../models/Account').model;
const uuid = require(‘uuid/v4');

function newAccount(request, reply) {
  let acc = new Account({
    accountId: uuid()
  });

  acc.save(err => {
    if (err) {
      console.error(err);

      return reply({
        status: 500,
        error: "Error saving"
      })
    }

    reply({
      successful: true,
      accountId: acc.accountId
    })
  });
}

module.exports = {
  newAccount
};
```

```
// route/handlers/accountHandlers.js

function newTransaction(request, reply) {
  Account.findOne({“accountId”: request.params.id})
    .exec()
    .then(acc => {
      if (acc) {
        acc.transaction.push({
          value: request.payload.value
        });

        acc.save(err => {
          if (err) {
            return reply({
              status: 500,
              error: "Error saving"
            })
          }

          reply({
            successful: true
          })
        })
      } else {
        // Not found
        reply({
          status: 404,
          error: "Account not found"
        });
      }
    });
}
```

App Engine Deploy

- App Engine requires an app.yaml to run
- Two lines required

```
runtime: nodejs
env: flex
```

https://cloud.google.com/sdk/docs/quickstarts

Deploying On App Engine

- gcloud app deploy

Important Notes

- App Engine does not run MongoDB locally
- Create an instance using Cloud Launcher from the GCP site
- Change the URL in your project to match the one you just created that's hosted in GCP

How To: Microservices

- You need
  - An API Gateway
  - Service discovery
- Good source for tools: Netflix OSS

Examples (Netflix OSS)

- API Gateway
  - Zuul
- Service Discovery
  - Eureka

Examples (Google Kubernetes)

- Service discovery built in (‘services’)
- API Gateway with Linkerd

Examples (Google App Engine)

- API gateway with Cloud Endpoint
- Service discovery using App Engine (essentially auto-scaling with load balancing)

Example (AWS)

- API Gateway
- Service discovery
  - Auto-scaling groups
  - Elastic Load Balancers