# Lecture 5: Push, REST & persistence

Olivier Liechti
TWEB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

# Agenda

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

| | | |
|---|---|---|
| 13h00 - 13h15 | 15’ | **Lecture**<br>Realtime Web & Socket.io |
| 13h15 - 13h30 | 15’ | **Lecture**<br>Persistence with mongoDB and mongoose |
| 13h30 - 14h00 | 30’ | **Lecture**<br>REST APIs (server and client side) |
| 14h00 - 18h00 | 240’ | **Project** |

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

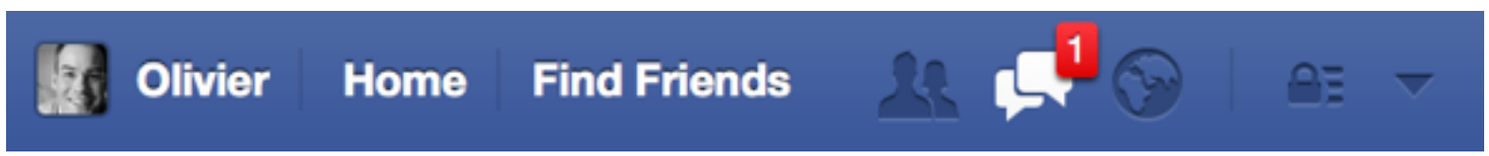
socket.io

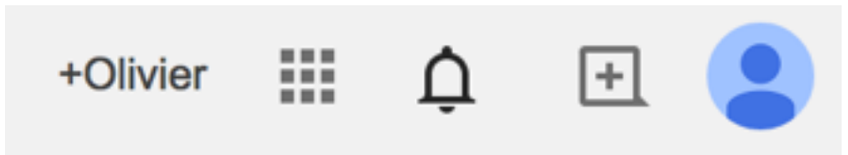
# Realtime Web with Socket.io

# The “realtime” web

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- The original architecture for the WWW is based on the **client-server model** and on the **request-reply messaging pattern**:
  - The client sends an HTTP request to the server.
  - The server sends an HTTP response back to the client.
  - They are done.
- In other words, the original architecture did not foresee the **need for the server to send notifications / messages** to the client side.
- Many applications do have this need. Over the years, **various techniques** have been proposed to address this need.



# Server-push techniques

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- The term "**Comet**" is used to categorize different techniques that allow a server to **push notifications** to the client.
- One idea (but which is **not efficient** and introduces **lag**) is to rely on **polling**. The client periodically sends a new AJAX request to ask the server: "do you have any notification for me?".
- Another idea is to **keep the underlying TCP connection open**, by **pretending that the server has not sent a full response yet**. In other words, the notifications sent by the server are fragments of the same HTTP response (it is a **streaming** technique).
- **Long polling** is a variation on these techniques. The client sends an HTTP request (to ask the server if a notification is available). The **server waits until a a notification is available** to send its response (which is positive). Not efficient, but solves the time lag issue.

http://cometdaily.com/2011/07/06/push-technology-comet-and-websockets-10-years-of-history-from-lightstreamers-perspective/

# WebSockets and HTML5

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- All of the previous techniques have raised **efficiency**, **reliability** and **browser-compatibility** issues.
- At the same time, the **need for bi-directional communication** has increased and is a requirement for many applications.
- For these reasons, a new protocol has been proposed and is becoming the standard way to implement bi-directional communication. This protocol has been named **WebSocket** and is specified in **RFC 6455**.
- It is **supported by recent versions of most popular web browsers** (**including mobile browsers**).
- Without going into the details, WebSocket specifies how a client and a server can use a **full-duplex** communication channel, after an initial **handshake**. The protocol also specifies how **framing** is handled.
- In addition to the **WebSocket protocol**, the **WebSocket API** has been specified as part of the HTML5 standardization effort.

# Using WebSockets in the browser

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Establish a connection

```
var exampleSocket = new WebSocket("ws://www.example.com/socketserver", "protocolOne");
```

Send data to the server

```
exampleSocket.onopen = function (event) {
  exampleSocket.send("Here's some text that the server is urgently awaiting!");
};
```

Process data sent by the server

```
exampleSocket.onmessage = function (event) {
  console.log(event.data);
}
```

# Using WebSockets with Node.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

There are several WebSocket modules. ws is one them.

```
var WebSocketServer = require('ws').Server, wss = new WebSocketServer({port: 8080});

wss.on('connection', function(ws) {
    ws.on('message', function(message) {
        console.log('received: %s', message);
    });
    ws.send('something');
});
```

http://einaros.github.io/ws/

# Socket.io

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- Although it is possible to use the WebSocket API and protocol directly, many developers use the **Socket.io library** as an alternative.
- A big benefit of Socket.io is that it supports several communication channels and mechanisms. Depending on the browser capabilities, it can **fall back** on older techniques (such as long polling).

```
var app = require('express').createServer();
var io = require('socket.io')(app);
app.listen(80);

app.get('/', function (req, res) {
  res.sendfile(__dirname + '/index.html');
});

io.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' });
  socket.on('my other event', function (data) {
    console.log(data);
  });
});
```

Server side (express.js)

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io.connect('http://localhost');
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('my other event', { my: 'data' });
  });
</script>
```

Client side

http://socket.io/docs/

# Using Socket.io with Angular.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- A **bower component** has been developed to facilitate the use of Socket.io from an Angular.js application: **angular-socket-io**.
- It is available on **Github**: https://github.com/btford/angular-socket-io.
- It is used by the Yeoman generator that we use for the project.

```
// in the top-level module of the app
angular.module('myApp', [
  'btford.socket-io',
  'myApp.MyCtrl'
]).
factory('mySocket', function (socketFactory) {
  return socketFactory();
}).
controller('MyCtrl', function (mySocket) {
  // ...
});
```

# Socket.io with the yeoman generator

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- Have a look at `client/app/main/main.controller.js`.
- This controller uses socket.io, so the authors have injected a dependency with the angular.js syntax shown below. You can use the same technique in your own controllers (simply add “socket” in your controller declaration).

dependency injection

```
angular.module('generatorAngularFullstackApp')
  .controller('MainCtrl', function ($scope, $http, socket) {
    $scope.awesomeThings = [];

    $http.get('/api/things').success(function(awesomeThings) {
      $scope.awesomeThings = awesomeThings;
      socket.syncUpdates('thing', $scope.awesomeThings);
    });
...
```

you can use socket.io via this variable; syncUpdates is an advanced function implemented in the generator, but you can use simpler socket.io functions instead.

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

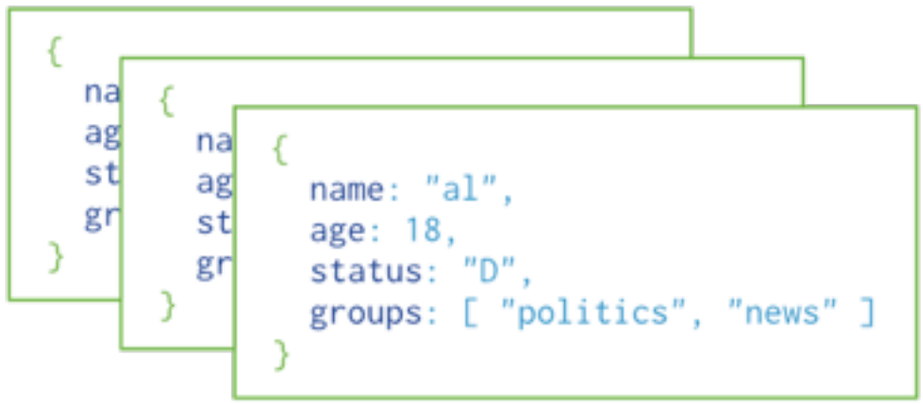


# Object ~~Relational~~ “Document” Mapping

# mongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- MongoDB is one of the most popular **NoSQL** databases (and one of the first to have been categorized as such).
- It is a **schema-less document-oriented** database:
  - The data store is made of several **collections**.
  - Every collection contains a set of **documents**, which you can think of as JSON objects.
  - The **structure of documents is not defined** *a priori* and is not enforced. This means that a collection can contain documents that have different fields.


http://docs.mongodb.org/manual/core/crud-introduction/

# Accessing mongoDB from Node.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- In the **Java ecosystem**, it is possible to interact with a RDBMS by using a JDBC driver:
  - The program loads the driver.
  - The program establishes a connection with the DB.
  - The program sends SQL queries to read and/or update the DB.
  - The program manipulates tabular result sets returned by the driver.
- With **Node.js and mongoDB**, the process is similar:
  - There is a **Node.js driver for mongoDB** (in fact, there are several).
  - A Node.js module can connect to a mongoDB server and issue queries to **manipulate collection and documents**.

# Example 1: connect and insert

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect("mongodb://localhost:27017/exampleDb", function(err, db) {
  if(err) { return console.dir(err); }

  var collection = db.collection('test');

  var doc1 = {'hello':'doc1'};
  var doc2 = {'hello':'doc2'};
  var lotsOfDocs = [{'hello':'doc3'}, {'hello':'doc4'}];

  collection.insert(doc1);
  collection.insert(doc2, {w:1}, function(err, result) {});
  collection.insert(lotsOfDocs, {w:1}, function(err, result) {});

});
```

http://mongodb.github.io/node-mongodb-native/api-articles/nodekoarticle1.html

# Example 2: query

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect("mongodb://localhost:27017/exampleDb", function(err, db) {
  if(err) { return console.dir(err); }

  var collection = db.collection('test');
  var docs = [{mykey:1}, {mykey:2}, {mykey:3}];
  collection.insert(docs, {w:1}, function(err, result) {

    // beware of memory consumption!
    collection.find().toArray(function(err, items) {});

    // better when many documents are returned
    var stream = collection.find({mykey:{$ne:2}}).stream();
    stream.on("data", function(item) {});
    stream.on("end", function() {});

    // special case when only one document is expected
    collection.findOne({mykey:1}, function(err, item) {});

  });
});
```

http://mongodb.github.io/node-mongodb-native/api-articles/nodekoarticle1.html

# mongoose: an ORM for MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

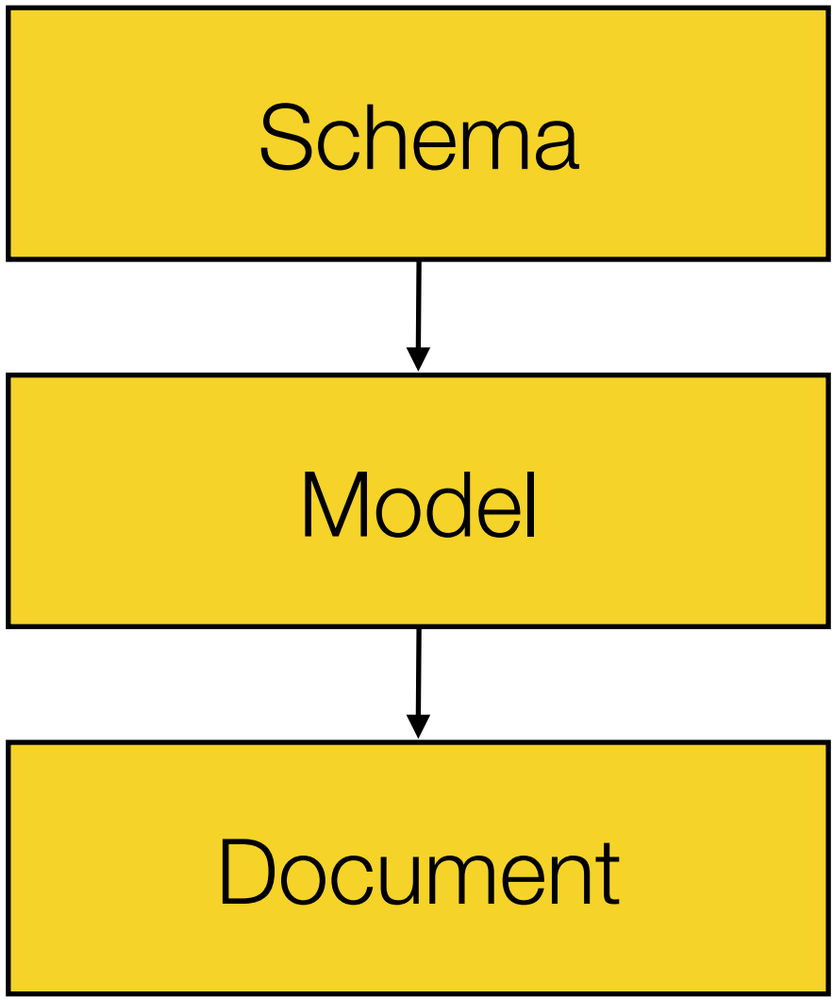
- In the **Java EE ecosystem**, we have seen how the **Java Persistence API** (JPA) specifies a standard way to interact with Object-Relational Mapping (ORM) frameworks**.**
  - The developer **first** creates an **object-oriented domain model**, by creating Entity classes and using various annotations (@Entity, @Id, @OneToMany, @Table, etc.)
  - He **then** uses an **Entity Manager** to **C**reate, **R**ead, **U**pdate and **D**elete objects in the DB**.**
  - The ORM framework takes care of the details: it **generates the schema** and the **SQL queries.**
- In the Javascript ecosystem, we have similar mechanisms**. With the particular yeoman generator that we use for the project**:
  - The authors have decided not use a relational database, but rather the **mongodb** document-oriented database**.**
  - They have decided to use **one of the data mapping tools** available for mongodb, namely **mongoose.** Since mongodb is a document-oriented database, it is more appropriate to talk about an Object-Document Mapping tool, rather than an ORM**.**

Why is not completely correct to say that mongoose is an ORM for MongoDB?

# mongoose: an ODM for MongoDB

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

*"Mongoose provides a **straight-forward**, schema-based solution to **modeling** your application data and includes built-in type **casting**, **validation**, **query** building, business logic hooks and more, out of the box."*




"Everything in Mongoose starts with a Schema. Each **schema maps to a MongoDB collection** and defines the shape of the documents within that collection."

"**Models are fancy constructors** compiled from our Schema definitions."

"Mongoose documents represent a **one-to-one mapping** to documents as stored in MongoDB. Each document is an **instance of its Model**."

http://mongoosejs.com/docs/guide.html

# Example

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

schema

model

collection

```
var userSchema = new mongoose.Schema({
  name: {
    first: String,
    last: { type: String, trim: true }
  },
  age: { type: Number, min: 0 }
});
```

```
var PUser = mongoose.model('PowerUsers', userSchema);

var johndoe = new PUser ({
  name: { first: 'John', last: '  Doe   ' },
  age: 25
});

johndoe.save(function (err) {if (err) console.log ('Error on save!')});
```

document

https://devcenter.heroku.com/articles/nodejs-mongoose

# Example: query

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

we can chain conditions

```
Person
.find({ occupation: /host/ })
.where('name.last').equals('Ghost')
.where('age').gt(17).lt(66)
.where('likes').in(['vaporizing', 'talking'])
.limit(10)
.sort('-occupation')
.select('name occupation')
.exec(callback);
```

we are interested in only some of the fields

we only want to get at most 10 documents

http://mongoosejs.com/docs/queries.html

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud


# RESTful APIs
# a **teaser** for an upcoming AMT lecture

# REST API

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- A REST API is **a type of Web Service interface** (and as such, is an alternative to a SOAP/WSDL API).
- A REST API is based on the core components of the Web architecture:
  - It exposes **resources** (a user, a photo, a sensor, a measure, a lecture)
  - It uses **URLs** to identify and locate resources
    - `/api/users`, `/api/users/8282`, `/api/users/8282/photos/`, etc.
  - It uses **HTTP methods** to expose **operations** applicable to resources
    - GET, POST, PUT, DELETE, PATCH
  - It uses **content negotiation** and **resource representation formats** to transfer **machine-understandable payloads** (json, xml, etc.)

# REST API

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud





R → 9.3
C → 9.5
U → 9.6
D → 9.7

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
GET /api/lectures/ HTTP/1.1
Accept: application/json
```

```
GET /api/lectures/238 HTTP/1.1
Accept: application/json
```

# C **READ** UD

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
GET /api/lectures/ HTTP/1.1
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-type: application/json

[
  {‘id’: ‘123’},
  {‘id’: ‘238’},
  {‘id’: ‘997’}
]
```

```
GET /api/lectures/238 HTTP/1.1
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-type: application/json

{
  ‘id’ : ‘238’,
  ‘title’ : ‘intro to mongodb’,
  ‘level’ : ‘beginner’
}
```

```
POST /api/lectures/ HTTP/1.1
Content-type: application/json

{
   ‘title’ : ‘intro to mongodb’,
   ‘level’ : ‘beginner’
}
```

```
PUT /api/lectures/238 HTTP/1.1
Content-type: application/json

{
   ‘title’ : ‘intro to MongoDB’,
   ‘level’ : ‘intermediate’
}
```

# CRU **DELETE**

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
DELETE /api/lectures/238 HTTP/1.1
```

Normal | Basic Auth | Digest Auth | OAuth 1.0 | No environment

http://localhost:9000/api/things/ GET URL params Headers (1)

Accept application/json Manage presets

Header Value

Send Preview Add to collection Reset

request

Body | Cookies (1) | Headers (8) STATUS 200 OK TIME 112 ms

response

Pretty | Raw | Preview JSON | XML


```
[
    {
        "_id": "545630336aae96b6be2ab0b3",
        "name": "Smart Build System",
        "info": "Build system ignores `spec` files, allowing you to keep tests alongside code. Automatic injection of scripts and styles into your index.html",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b1",
        "name": "Development Tools",
        "info": "Integration with popular tools such as Bower, Grunt, Karma, Mocha, JSHint, Node Inspector, Livereload, Protractor, Jade, Stylus, Sass, CoffeeScript, and Less.",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b4",
        "name": "Modular Structure",
        "info": "Best practice client and server structures allow for more code reusability and maximum scalability",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b2",
        "name": "Server and Client integration",
        "info": "Built with a powerful and fun stack: MongoDB, Express, AngularJS, and Node.",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b5",
        "name": "Optimized Build",
        "info": "Build process packs up your templates as a single JavaScript payload, minifies your scripts/css/images, and rewrites asset names for caching.",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b6",
        "name": "Deployment Ready",
        "info": "Easily deploy your app to Heroku or Openshift with the heroku and openshift subgenerators",
        "__v": 0
    }
]
```

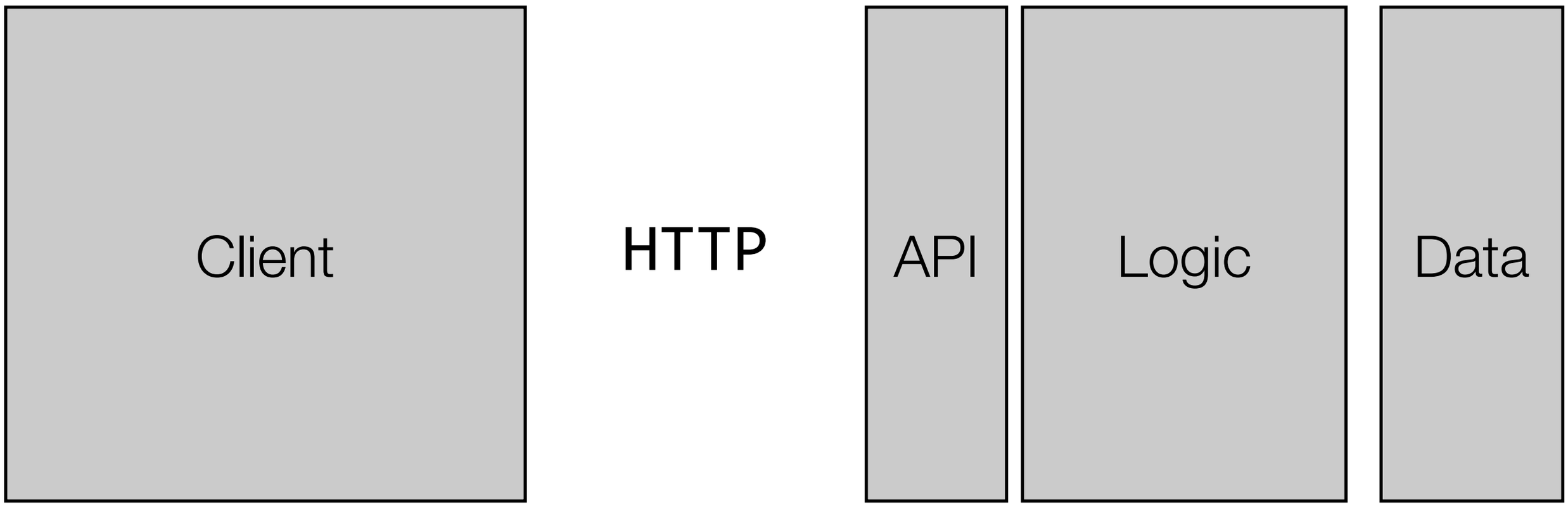

heig-vd

Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

# REST API

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

## GET/POST/PUT/DELETE



heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

YEOMAN

# Creating a REST API with the “fullstack” yeoman generator (and express)

# Create an API endpoint on the server

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- Our favorite yeoman generator provides various "**sub-generators**" to add application components (both on the server and on the client side).
- The **endpoint sub-generator** makes it easy to add REST endpoint. It takes care of the **express.js routing** and generates **mongoose** code that we can use as a starting point.

```
$ yo angular-fullstack:endpoint fruit
? What will the url of your endpoint to be? /api/fruits
   create server/api/fruit/index.js
   create server/api/fruit/fruit.controller.js
   create server/api/fruit/fruit.model.js
   create server/api/fruit/fruit.socket.js
   create server/api/fruit/fruit.spec.js
```

# api/fruit/index.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
'use strict';

var express = require('express');
var controller = require('./fruit.controller');

var router = express.Router();

router.get('/', controller.index);
router.get('/:id', controller.show);
router.post('/', controller.create);
router.put('/:id', controller.update);
router.patch('/:id', controller.update);
router.delete('/:id', controller.destroy);

module.exports = router;
```

*In most cases, we can use this “as is”*

# api/fruit/fruit.model.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
'use strict';

var mongoose = require('mongoose'),
    Schema = mongoose.Schema;

var FruitSchema = new Schema({
  name: String,
  info: String,
  active: Boolean
});

module.exports = mongoose.model('Fruit', FruitSchema);
```

*This is obviously one script we need to change - the generator has no way to know the structure of our domain objects*

# api/fruit/fruit.controller.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
'use strict';

var _ = require('lodash');
var Fruit = require('./fruit.model');

// Get list of fruits
exports.index = function(req, res) {
  Fruit.find(function (err, fruits) {
    if(err) { return handleError(res, err); }
    return res.json(200, fruits);
  });
};

// Get a single fruit
exports.show = function(req, res) {
  Fruit.findById(req.params.id, function (err, fruit) {
    if(err) { return handleError(res, err); }
    if(!fruit) { return res.send(404); }
    return res.json(fruit);
  });
};

// Creates a new fruit in the DB.
exports.create = function(req, res) {
  Fruit.create(req.body, function(err, fruit) {
    if(err) { return handleError(res, err); }
    return res.json(201, fruit);
  });
};

// Updates an existing fruit in the DB.
exports.update = function(req, res) {
  if(req.body._id) { delete req.body._id; }
  Fruit.findById(req.params.id, function (err, fruit) {
    if (err) { return handleError(res, err); }
    if(!fruit) { return res.send(404); }
    var updated = _.merge(fruit, req.body);
    updated.save(function (err) {
      if (err) { return handleError(res, err); }
      return res.json(200, fruit);
    });
  });
};

// Deletes a fruit from the DB.
exports.destroy = function(req, res) {
  Fruit.findById(req.params.id, function (err, fruit) {
    if(err) { return handleError(res, err); }
    if(!fruit) { return res.send(404); }
    fruit.remove(function(err) {
      if(err) { return handleError(res, err); }
      return res.send(204);
    });
  });
};

function handleError(res, err) {
  return res.send(500, err);
}
```

***Use mongoose to interact with DB***

***these functions are called in the express.js router***

*In most cases, we can use this "as is"*

# api/fruit/fruit.socket.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
/**
 * Broadcast updates to client when the model changes
 */
'use strict';
var Fruit = require('./fruit.model');

exports.register = function(socket) {
  Fruit.schema.post('save', function (doc) {
    onSave(socket, doc);
  });
  Fruit.schema.post('remove', function (doc) {
    onRemove(socket, doc);
  });
}

function onSave(socket, doc, cb) {
  socket.emit('fruit:save', doc);
}

function onRemove(socket, doc, cb) {
  socket.emit('fruit:remove', doc);
}
```

***Register a hook to be notified after (post) "save" and "remove" function calls***

***Push an event notification to connected clients, with socket.io***

*This is not typical for most REST APIs (which implement the standard request-reply messaging pattern).*

# config/socketio.js

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
/**
 * Socket.io configuration
 */

'use strict';

var config = require('./environment');

// When the user disconnects.. perform this
function onDisconnect(socket) {
}

// When the user connects.. perform this
function onConnect(socket) {
  // When the client emits 'info', this listens and executes
  socket.on('info', function (data) {
    console.info('[%s] %s', socket.address,
JSON.stringify(data, null, 2));
  });

  // Insert sockets below
  require('../api/fruit/fruit.socket').register(socket);
  require('../api/lecture/lecture.socket').register(socket);
  require('../api/thing/thing.socket').register(socket);
}
```

```
module.exports = function (socketio) {
  socketio.on('connection', function (socket) {
    socket.address = socket.handshake.address !== null ?
            socket.handshake.address.address + ':' +
socket.handshake.address.port :
            process.env.DOMAIN;

    socket.connectedAt = new Date();

    // Call onDisconnect.
    socket.on('disconnect', function () {
      onDisconnect(socket);
      console.info('[%s] DISCONNECTED', socket.address);
    });

    // Call onConnect.
    onConnect(socket);
    console.info('[%s] CONNECTED', socket.address);
  });
};
```

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

$http

- $httpProvider
- service in module ng

$resource

- service in module ngResource


# Calling a REST API from Angular.js

# $http

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- "The `$http` service is a **core Angular service** that facilitates communication with the **remote HTTP servers** via the browser's **XMLHttpRequest** object or via JSONP."
- In other words: it makes it easier for you to make **AJAX calls**.
- In other words: it provides a first mechanism to invoke **REST APIs** (fairly low-level)

```
// Simple GET request example :
$http.get('/someUrl').
  success(function(data, status, headers, config) {
    // this callback will be called asynchronously
    // when the response is available
  }).
  error(function(data, status, headers, config) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

https://docs.angularjs.org/api/ng/service/$http

# Usage in the angular fullstack generator

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- In the `main.controller.js` script, `$http` is used to interact with the Things REST API (the "things" are the notes added on the front page).

```
angular.module('generatorAngularFullstackApp')
  .controller('MainCtrl', function ($scope, $http, socket) {
    $scope.awesomeThings = [];

    $http.get('/api/things').success(function(awesomeThings) {
      $scope.awesomeThings = awesomeThings;
      socket.syncUpdates('thing', $scope.awesomeThings);
    });

    $scope.addThing = function() {
      if($scope.newThing === '') {
        return;
      }
      $http.post('/api/things', { name: $scope.newThing });
      $scope.newThing = '';
    };
...
  });
```

*dependency injection*

*HTTP GET to retrieve things and update the view (via scope)*

*HTTP POST to create a new thing based on the form data (via scope)*

# $resource

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- “A factory which creates a resource object that lets you interact with RESTful server-side data sources. The returned resource object has action methods which provide high-level behaviors without the need to interact with the low level $http service.”

```
var User = $resource('/user/:userId', {userId:'@id'});
var user = User.get({userId:123}, function() {
  user.abc = true;
  user.$save();
});
```

# Usage in the angular fullstack generator

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- In the `user.service.js` script, `$resource` is used to facilitate the interaction with the user REST API. The resource created by the factory is used in the `auth.service.js` script.

```
angular.module('generatorAngularFullstackApp')
  .controller('MainCtrl', function ($scope, $http, socket) {
    $scope.awesomeThings = [];

    $http.get('/api/things').success(function(awesomeThings) {
      $scope.awesomeThings = awesomeThings;
      socket.syncUpdates('thing', $scope.awesomeThings);
    });

    $scope.addThing = function() {
      if($scope.newThing === '') {
        return;
      }
      $http.post('/api/things', { name: $scope.newThing });
      $scope.newThing = '';
    };
...
  });
```

*dependency injection*

*HTTP GET to retrieve things and update the view (via scope)*

*HTTP POST to create a new thing based on the form data (via scope)*

# Recommended tutorial

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- "**Creating a CRUD App in Minutes with Angular's $resource**" (Sandeep Panda, June 11th 2014)
- http://www.sitepoint.com/creating-crud-app-minutes-angulars-resource/

```
angular.module('movieApp.services', []).factory('Movie', function($resource) {
  return $resource('http://movieapp-sitepointdemos.rhcloud.com/api/movies/:id', { id: '@_id' }, {
    update: {
      method: 'PUT'
    }
  });
});
```

*dependency injection*

```
angular.module('movieApp.controllers', []).controller('MovieListController', function($scope,
$state, popupService, $window, Movie) {
  $scope.movies = Movie.query(); //fetch all movies. Issues a GET to /api/movies

  $scope.deleteMovie = function(movie) { // Delete a movie. Issues a DELETE to /api/movies/:id
    if (popupService.showPopup('Really delete this?')) {
      movie.$delete(function() {
        $window.location.href = ''; //redirect to home
      });
    }
  };
});
...
```