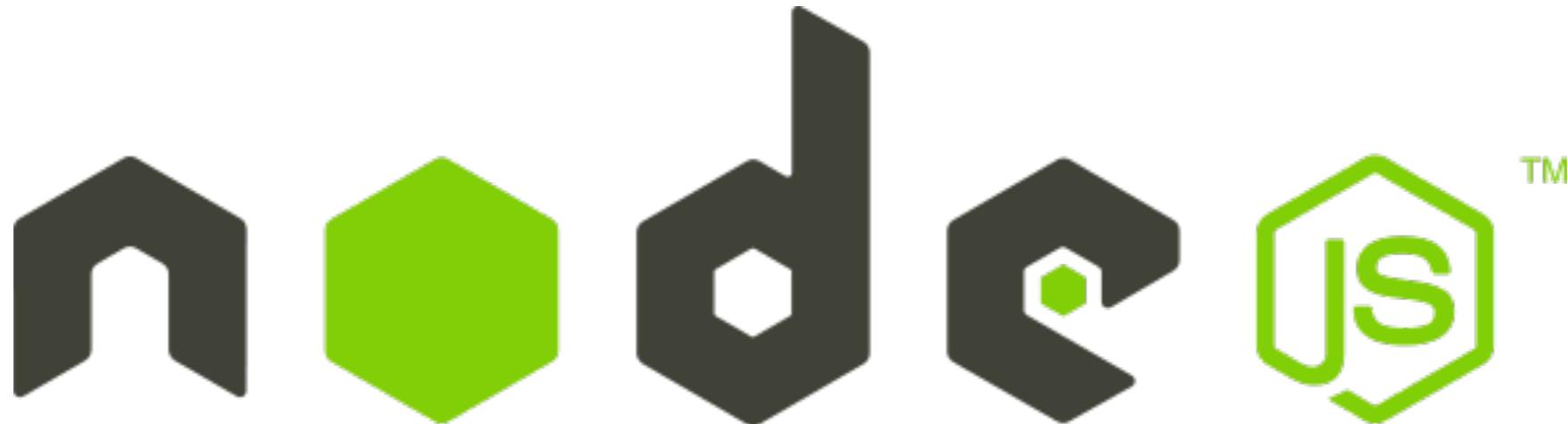


REST APIs with Express.js

Olivier Liechti & Simon Oulevay
COMEM Web Services 2016

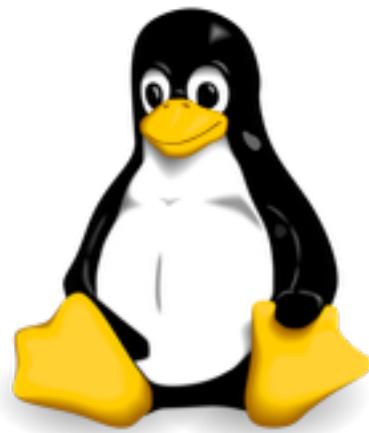


Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Install Node.js

Install a real command line



or



?

Good to go!

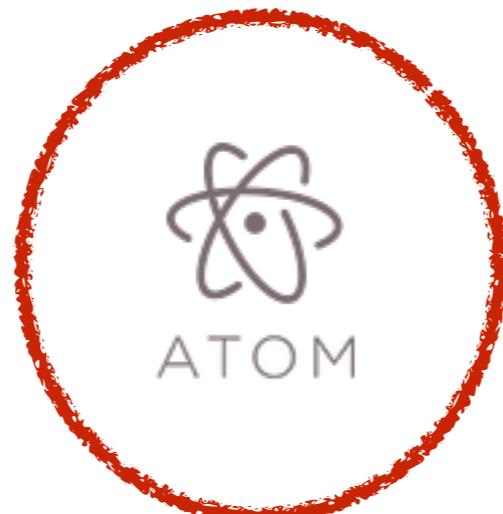


<http://babun.github.io>

Install a real code editor (not Notepad)



TextMate



Brackets



Sublime Text



Komodo IDE



Cloud9 IDE



NetBeans



Install Node.js & npm



<https://nodejs.org/>

Setup #4

Make it work!

```
$> node -v  
4.3.1  
  
$> node  
> 1 + 2  
3
```

- Open your command line.
- Type **node** and press enter. A new prompt should appear, indicating that you are in the Node.js console.
- Type **1 + 2** and press enter. Node.js should give you the result.
- (Type **0.1 + 0.2** if you want to see that your CPU can't count.)

heig-vd

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



YEOMAN

Scaffolding with Yeoman



How do I **bootstrap** and **structure** my project?

- Based on the specifications, we know that we will develop a REST API with Express, but we'll also use MongoDB and Mongoose, and probably other tools.
- What should we do? **Start from scratch** or use some kind of **skeleton**? What are our **options**? What are the **professional** front-end developers doing?

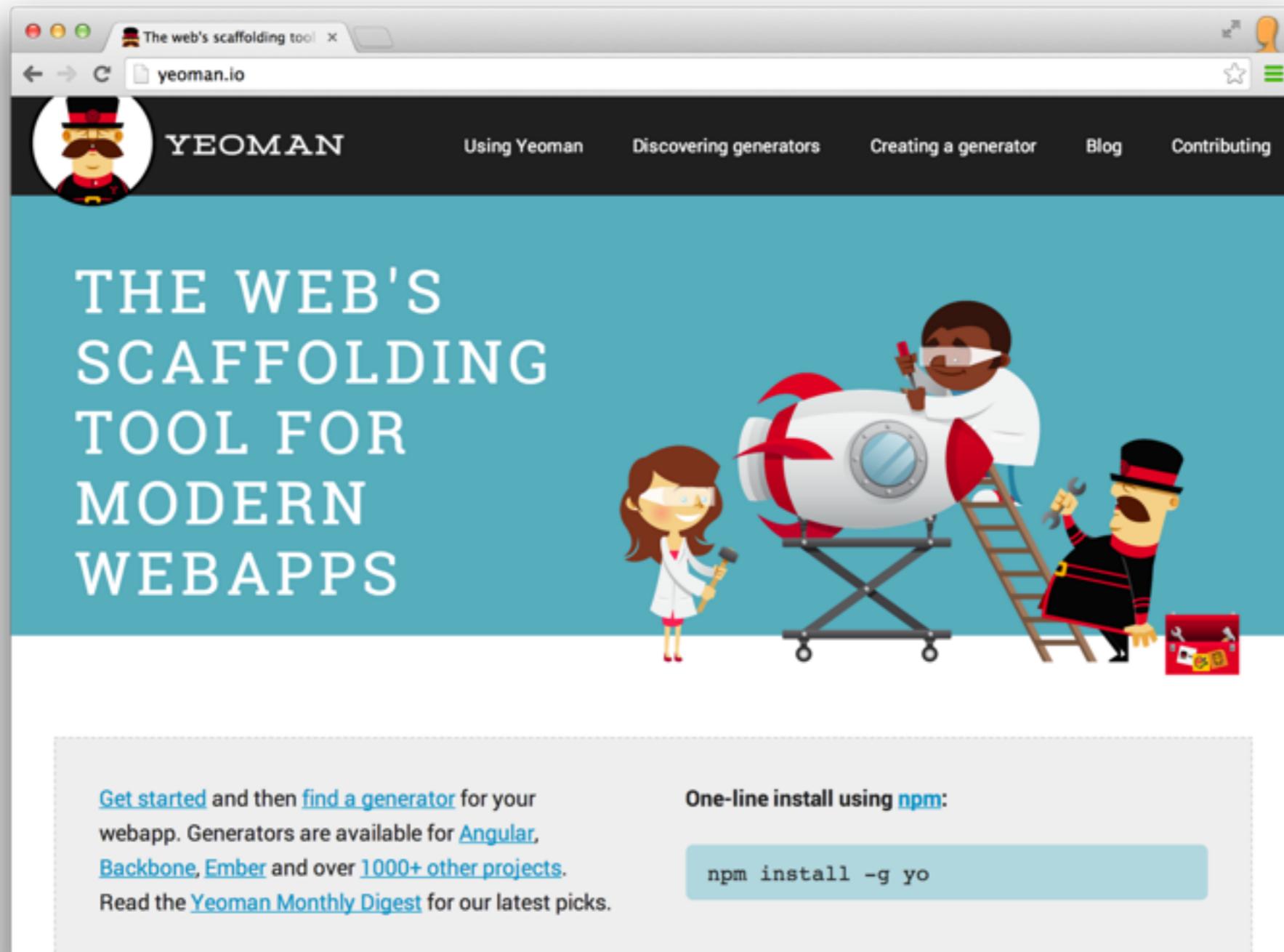


[Paul Irish, "Delivering the goods" - Fluent 2014 Keynote](#)
by O'Reilly · 7 months ago · 29,621 views
Fluent 2014, "Keynote With Paul Irish". About Paul Irish (Google): Paul Irish is a front-end developer who loves the web. He is on ...
[HD](#)



[Fluent 2013: Paul Irish, "JavaScript Authoring Tooling"](#)
by O'Reilly · 1 year ago · 35,810 views
<http://fluentconf.com> To view a complete archive of the Fluent 2013 tutorials and sessions, check out the All Access video ...
[HD](#)

Meet Yeoman



What is Yeoman?

- Yeoman is a **combination of tools**, which allows to you to setup a **complete, automated, efficient and reliable development workflow**.
- **Yo** is a tool for generating project skeletons (**scaffolding**). You can create and share your skeletons. **Yo generators are npm modules** and you can find one for most popular web frameworks.
- **Bower** is a tool for managing “**web dependencies**”. Not only javascript modules, but also CSS files, images, etc.
- **Grunt** is a **task runner**. It is the tool that drives your automated process, by executing a series of tasks. There are lots of grunt plugins provided by the community for all aspects of your project.



YO



GRUNT



BOWER

What is Yeoman?

heig-vd

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

Generators | Yeoman

yeoman.io/generators/

YEOMAN

Using Yeoman Discovering generators Creating a generator Blog Contributing

GENERATORS

Your generator must have the "yeoman-generator" keyword and a repo description to be listed. Official generators are marked with

express

Name	Description	Author	Stars
angular-fullstack	AngularJS with an Express server	Tyler Henkel	2788
express	An express generator for Yeoman, based on the express command line tool	petecoop	245
mean-seed	MEAN Seed / MEAN Stack (AngularJS, node.js app) - MongoDB, Express, Angular, Node, Grunt, Bower, Yo. 'Core' and 'Module' subgenerators for customization outside of that	Luke Madera	113
yoosurt	A "Choose your own adventure" generator for creating static sites and single page applications. Helps you harness the power of your	Jake Larson	100

Install Yeoman



YEOMAN

<http://yeoman.io/learning/>

```
$> sudo npm install -g npm
$> sudo npm install -g bower grunt-cli yo
```



How do you **pick a generator** for your project?

- You probably **have an idea of the framework(s) you want to use** on the server and or client side (express, angular, backbone, etc.). You will use this as a first filter.
- Some of the generators are **supported by the Yeoman Team**. That is probably a good indication about the quality and support over time (evolution).
- Developers who use generators can “**star**” those they like. **Sorting by popularity** is also an interesting indication. If the community is big, you can expect issues to be reported and fixed, to see new features, etc.
- After you have identified **promising candidates**, you need to get a **first impression**. Generate and build a project with each candidate. Look at their Github repository. Do you like what you see? Do you like the documentation?
- Often, you will need to choose between “**lightweight**” and very “**rich**” generators. Lightweight generators are easier to learn and give you more control (but more work). Rich generators do a lot of things out-of-the-box but can be intimidating at first (learning curve to understand the skeleton).

express
web application
framework for
node

Express.js

Express.js: core functionality

- **Implementation of the Model View Controller (MVC) pattern**
- **Routing**
 - **Mapping** between HTTP request attributes (URL, method, etc.) and **controllers** (handlers)
- **Middleware**
 - **Interception** and possibly **transformation** of HTTP requests and responses during their processing.
- **Template engines**
 - Rendering of views with **pluggable template engines**.



Meet the Yeoman **express** generator.

The screenshot shows a web browser window with the URL <https://github.com/petecoop/generator-express>. The page content includes a section titled "Features" with a bulleted list of capabilities, and a "Getting started" section with instructions for installation and setup.

Features

- Basic or MVC style file structure
- CoffeeScript Support
- Gulp or Grunt build tools with file watching and livereload
- .editorconfig for consistent coding styles within text editors
- Support View engines:
 - Jade
 - Handlebars
 - Swig
 - EJS
 - Marko
 - Nunjucks
- Supported CSS pre-processors
 - SASS (both node-sass and ruby sass)
 - LESS
 - Stylus
- Supported Databases (with MVC structure):
 - MongoDB
 - MySQL
 - PostgreSQL
 - RethinkDB
 - SQLite

Getting started

- Make sure you have `yo` installed: `npm install -g yo`
- Install the generator **globally**: `npm install -g generator-express`
- Run: `yo express` and select Basic. Add `--coffee` if you require CoffeeScript.
- Run: `grunt` or `gulp` to run the local server at `localhost:3000`, the grunt/gulp tasks include live reloading for views, css in public/css and restarting the server for changes to app.js or js in routes/



Why and when is this generator interesting?

- It is **more powerful** than the standard generator provided by the Express.js framework.
- It allows you to easily integrate **persistence**, with a choice of several relational (e.g. MySQL) and NoSQL (e.g. MongoDB) data stores. In that sense, it generates a full multi-tiered MVC application.
- It comes with a **Grunt file**, which includes the **live reload plugin**.
- The generated skeleton and Grunt file are rather **lightweight and straightforward**. It is fairly easy to get into the generated code and to add your application-specific code.

Install the Yeoman Express generator

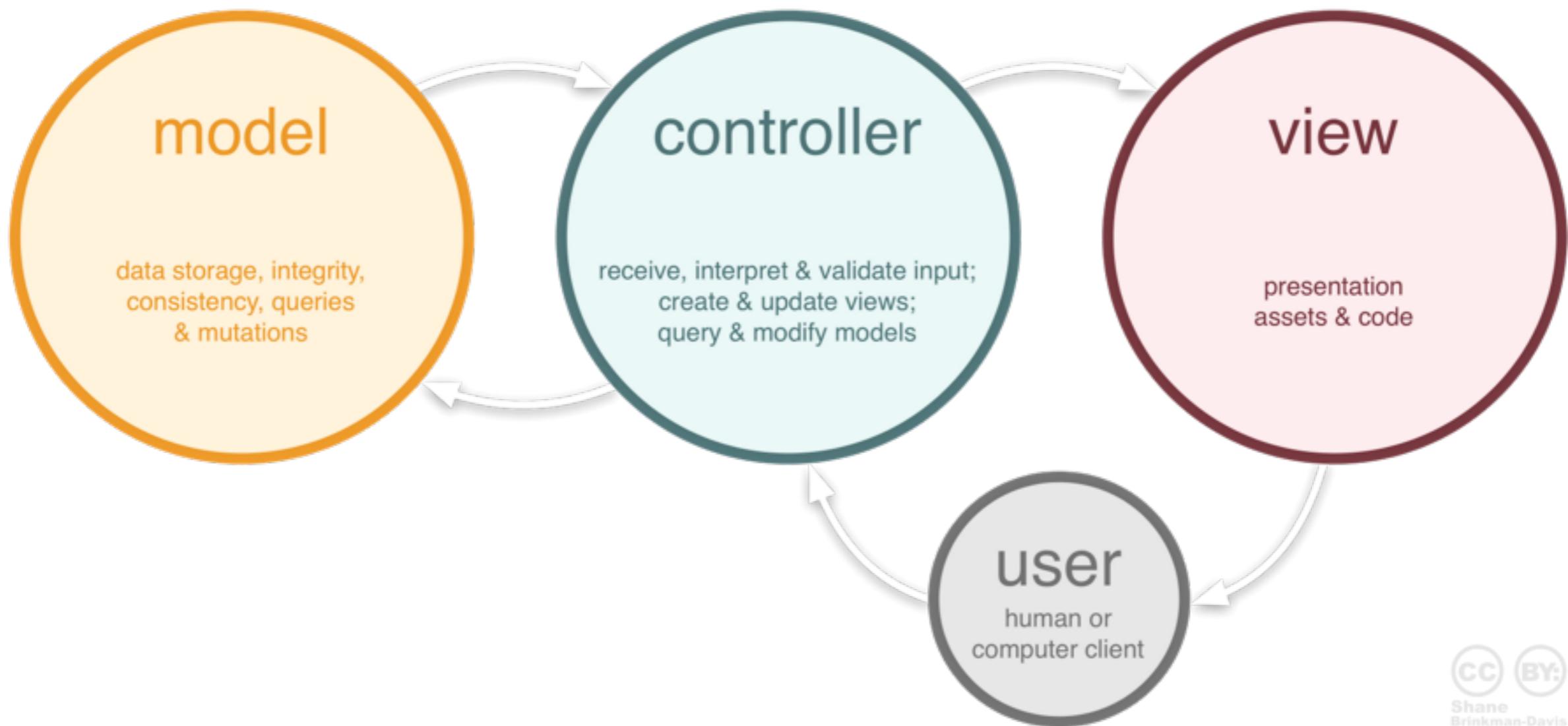
express

```
$> npm install -g generator-express  
$> cd /path/to/project
```

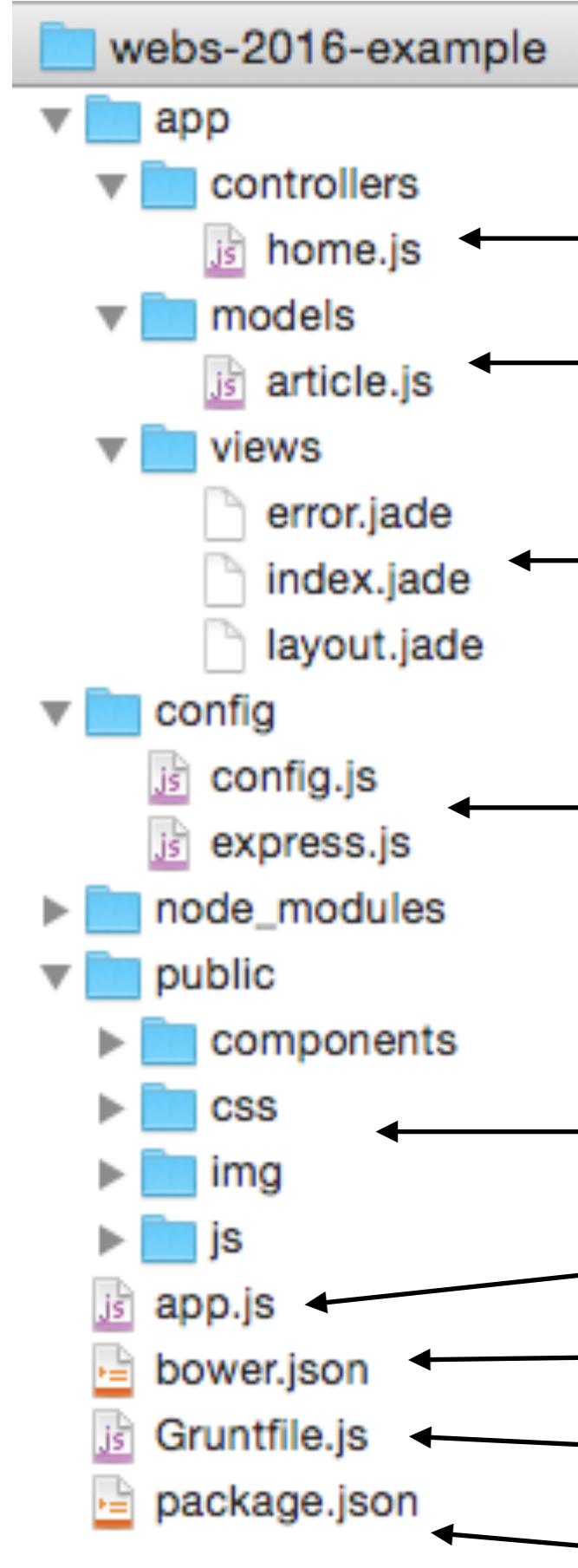
```
$> yo express
? Would you like to create a new directory for your project? No
? Select a version to install: MVC
? Select a view engine to use: Jade
? Select a css preprocessor to use (Sass Requires Ruby): Stylus
? Select a database to use: MongoDB
? Select a build tool to use: Grunt
  create bower.json
  create package.json
  create .bowerrc
  create .editorconfig
  create .gitignore
  create app.js
  create app/controllers/home.js
  create app/models/article.js
  create config/config.js
  create config/express.js
  create app/views/error.jade
  create app/views/index.jade
  create app/views/layout.jade
  create public/css/style.styl
  create Gruntfile.js
```

I'm all done. Running npm install & bower install for you to install the required dependencies. If this fails, try running the command yourself.

Express uses a **Model-View-Controller (MVC)** structure



Express File Structure



controller (HTML view or API resource)

model (data access with Mongoose)

view (Jade templates compiled to HTML)

configuration

client-side assets (CSS, JS, images, etc)

entry point

bower package definition

tasks configuration

npm package definition

Express Entry Point: app.js

dependencies

```
var express = require('express'),  
config = require('./config/config'),  
glob = require('glob'),  
mongoose = require('mongoose');  
  
mongoose.connect(config.db);  
var db = mongoose.connection;  
db.on('error', function () {  
  throw new Error('unable to connect to database at ' + config.db);  
});  
  
var models = glob.sync(config.root + '/app/models/*.js');  
models.forEach(function (model) {  
  require(model);  
});  
var app = express();  
  
require('./config/express')(app, config);  
  
app.listen(config.port, function () {  
  console.log('Express server listening on port ' + config.port);  
});
```

npm packages
local modules

heig-vd

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

database
connection

asynchronous
callback on event

models
configuration

start the server

asynchronous
callback function

Your Express app's package.json:

```
{  
  "name": "webs-2016-example",  
  "version": "0.0.1",  
  "private": true,  
  "scripts": {  
    "start": "node app.js" ← how to start it  
  },  
  "dependencies": {  
    "express": "^4.13.3", ← the express package  
    "serve-favicon": "^2.3.0",  
    "morgan": "^1.6.1",  
    "cookie-parser": "^1.3.3",  
    "body-parser": "^1.13.3", ← express middleware packages  
    "compression": "^1.5.2",  
    "method-override": "^2.3.0",  
    "glob": "^6.0.4",  
    "mongoose": "^4.1.2", ← other dependencies  
    "jade": "^1.11.0"  
  },  
  "devDependencies": {  
    "grunt": "^0.4.5", ← development tools (not  
    "grunt-develop": "^0.4.0", used in production)  
    "grunt-contrib-stylus": "^1.0.0",  
    "grunt-contrib-watch": "^0.6.1",  
    "request": "^2.60.0",  
    "time-grunt": "^1.2.1",  
    "load-grunt-tasks": "^3.2.0"  
  }  
}
```

express
web application
framework for
node

Express.js Middleware

Express.js: middleware (filters)

- Incoming HTTP requests can be processed by multiple components, organized in a pipeline
- The components can inspect and even modify the incoming HTTP requests and HTTP responses (think about security, compression, etc.).
- Express.js calls these components "**middleware**" functions
- Middleware can be chained. They can intercept requests at different levels (all requests, requests under a certain path, requests handled by a specific router, etc.)
- **Built-in middleware** components are available (in separate npm modules).
- We will use the **express.static** middleware to serve static content (e.g. HTML)

Using middleware

Express is a routing and middleware web framework with minimal functionality of its own: An Express application is essentially a series of middleware calls.

Middleware is a function with access to the `request object` (`req`), the `response object` (`res`), and the next middleware in the application's request-response cycle, commonly denoted by a variable named `next`.

Middleware can:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware in the stack.

If the current middleware does not end the request-response cycle, it must call `next()` to pass control to the next middleware, otherwise the request will be left hanging.

An Express application can use the following kinds of middleware:

- [Application-level middleware](#)
- [Router-level middleware](#)
- [Error-handling middleware](#)
- [Built-in middleware](#)
- [Third-party middleware](#)

You can load application-level and router-level middleware with an optional mount path. Also, you can load a series of middleware functions together, creating a sub-stack of the middleware system at a mount point.

Express Configuration: config/config.js

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

var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var compress = require('compression');
var methodOverride = require('method-override');

module.exports = function(app, config) {
  var env = process.env.NODE_ENV || 'development';
  app.locals.ENV = env;
  app.locals.ENV_DEVELOPMENT = env == 'development';

  app.set('views', config.root + '/app/views');
  app.set('view engine', 'jade');

  // app.use(favicon(config.root + '/public/img/favicon.ico'));
  app.use(logger('dev'));
  app.use(bodyParser.json());
  app.use(bodyParser.urlencoded({
    extended: true
  }));
  app.use(cookieParser());
  app.use(compress());
  app.use(express.static(config.root + '/public'));
  app.use(methodOverride());
```

third-party
middleware
packages

use the middleware
one after the other;
each can modify the
request, send a
response, or pass the
request along to the
next middleware

From the API reference

Application-level middleware

Bind application-level middleware to an instance of the [app object](#) with `app.use()` and `app.METHOD()`, where `METHOD` is the HTTP method of the request that it handles, such as `GET`, `PUT`, `POST`, and so on, in lowercase. For example:

```
var app = express();

// a middleware with no mount path; gets executed for every request to the app
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});

// a middleware mounted on /user/:id; will be executed for any type of HTTP request to /user/:id
app.use('/user/:id', function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});

// a route and its handler function (middleware system) which handles GET requests to /user/:id
app.get('/user/:id', function (req, res, next) {
  res.send('USER');
});
```

Adding your own middleware

```
...  
  
app.use(express.static(config.root + '/public'));  
app.use(methodOverride());  
  
var controllers = glob.sync(config.root + '/app/controllers/*.js');  
controllers.forEach(function (controller) {  
    require(controller)(app);  
});  
  
app.use(function (req, res, next) {  
    var err = new Error('Not Found');  
    err.status = 404;  
    next(err);  
});  
  
// ...  
  
app.use(function (err, req, res, next) {  
    res.status(err.status || 500);  
    res.render('error', {  
        message: err.message,  
        error: {},  
        title: 'error'  
    });  
});
```

middleware packages

controllers used as middleware

last middleware that will catch requests not already handled

error-handling middleware

express
web application
framework for
node

Express.js Routing

Express.js: server-side routing

- **Routing** consists in finding some piece of code (a function) to execute when an HTTP request has been issued.
- We will see later (in a few weeks) that routing can happen on the client side. Today, we are looking at **routing on the server side**.
- Routing is part of the typical **Model-View-Controller (MVC) pattern** implemented by web frameworks (not only in JavaScript, but also in other languages).
- **Routing consists in finding the right controller when a request comes in.**
- The controller will then get a model and delegate the rendering of a view to a template engine.

The home page

In app/controllers/home.js

```
var express = require('express'),  
  router = express.Router(), ← Express Router  
  mongoose = require('mongoose'),  
  Article = mongoose.model('Article');  
  
module.exports = function (app) {  
  app.use('/', router); ← Add the router as middleware  
};  
  
router.get('/', function (req, res, next) {  
  Article.find(function (err, articles) {  
    if (err) return next(err);  
    res.render('index', {  
      title: 'Generator-Express MVC',  
      articles: articles  
    });  
  });  
});
```

Express Router

Add the router as middleware
(filtered by path)

Define a route: GET /

Routing

Routing refers to the definition of end points (URIs) to an application and how it responds to client requests.

A route is a combination of a URI, a HTTP request method (GET, POST, and so on), and one or more handlers for the endpoint. It takes the following structure `app.METHOD(path, [callback...], callback)`, where `app` is an instance of `express`, `METHOD` is an [HTTP request method](#), `path` is a path on the server, and `callback` is the function executed when the route is matched.

The following is an example of a very basic route.

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

// respond with "hello world" when a GET request is made to the homepage
app.get('/', function(req, res) {
  res.send('hello world');
});
```

- Route **methods** (GET, POST, PUT)
- Route **paths** ('/', '/home', '/students')
- **Request and response objects**

Router

A `router` object is an isolated instance of middleware and routes. You can think of it as a “mini-application,” capable only of performing middleware and routing functions. Every Express application has a built-in app router.

A router behaves like middleware itself, so you can use it as an argument to `app.use()` or as the argument to another router’s `use()` method.

The top-level `express` object has a `Router()` function that creates a new `router` object.

`Router([options])`

Create a new router as follows:

```
var router = express.Router([options]);
```

- For large applications, it is better to split the controllers in multiple, isolated components. You should use multiple routers for that purpose.
- `express.Router()` creates a new router. (no need for `r = new Router()`)

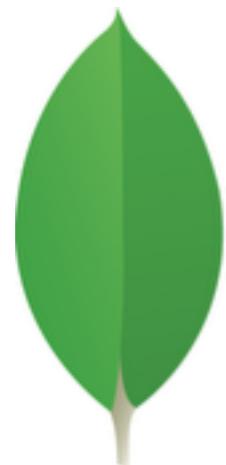
Express.js resources

Documentation

<http://expressjs.com/en/4x/api.html>

heig-vd

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



mongoDB®

MongoDB

NoSQL database



Cassandra

mongoDB



membase

 **riak**

document-oriented
database

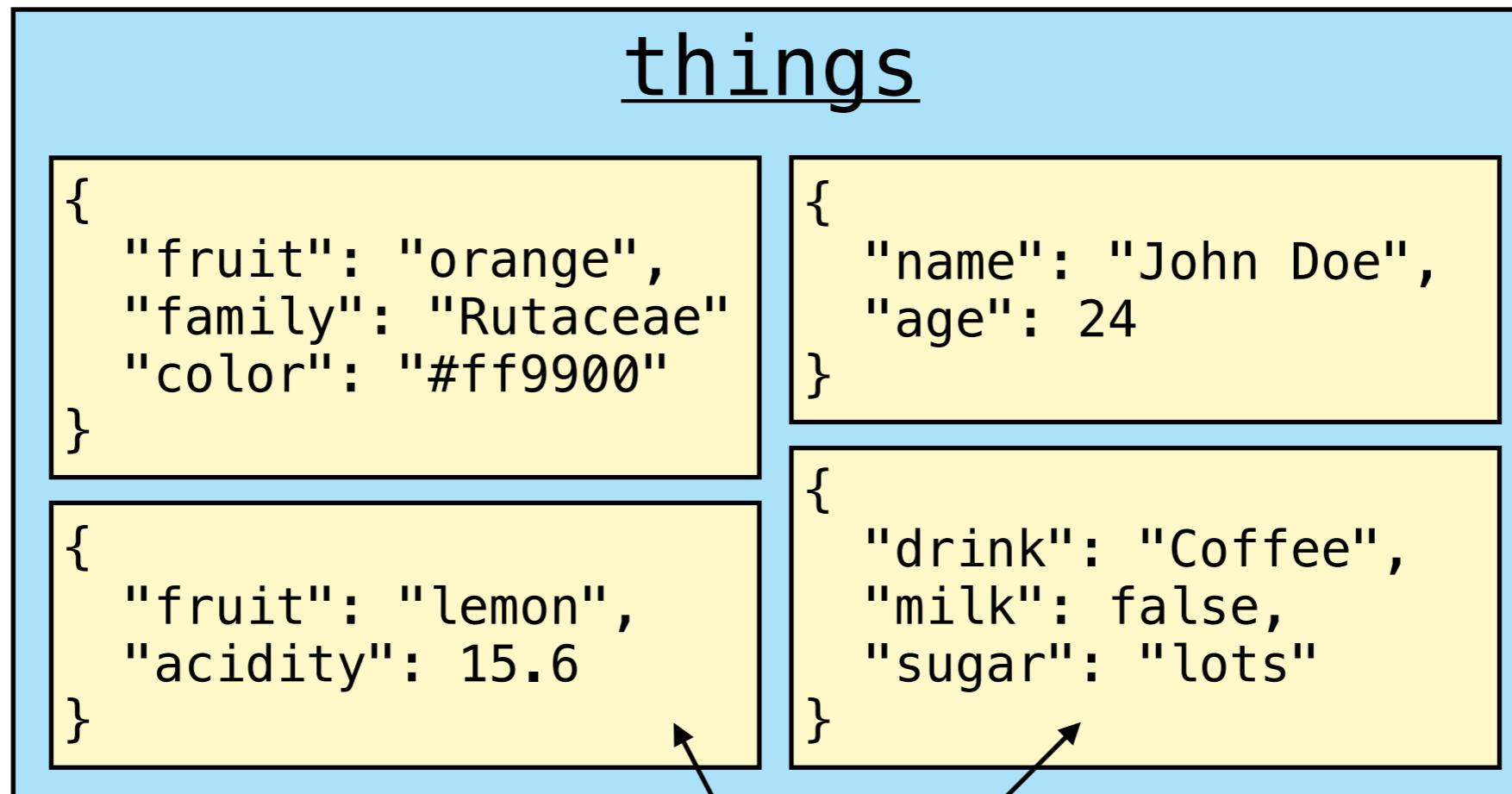
implementation validation agile protocol
textslots optimized standard hierarchies unknown
information query semi rows collections
relational string triggers schema designed empty concurrent self lookup similar explicitly offer language
empty concurrent self lookup similar explicitly offer language
similar explicitly offer language transient
rowgrouping rowgrouping rowgrouping rowgrouping
binary database sailing
retrieval statistics retains term rigid differs structured
encodings license clustering formats
field data encode ways notions simple
versioning encapsulate document structural relevance retrieve
semantics

Store **JSON**-like documents

```
{  
  "_id" : ObjectId("54c955492b7c8eb21818bd09"),  
  "address" : {  
    "street" : "2 Avenue",  
    "zipcode" : "10075",  
    "building" : "1480",  
    "coord" : [ -73.9557413, 40.7720266 ],  
  },  
  "borough" : "Manhattan",  
  "cuisine" : "Italian",  
  "grades" : [  
    {  
      "date" : ISODate("2014-10-01T00:00:00Z"),  
      "grade" : "A",  
      "score" : 11  
    },  
    {  
      "date" : ISODate("2014-01-16T00:00:00Z"),  
      "grade" : "B",  
      "score" : 17  
    }  
  "name" : "Vella",  
  "restaurant_id" : "41704620"  
}
```

Schema-less collections

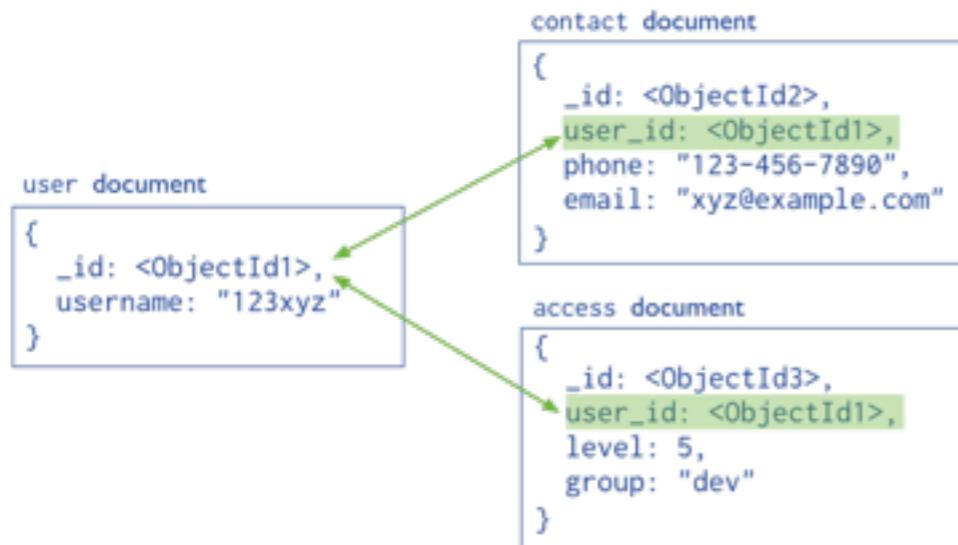
Collection (~ SQL table, but no schema)



Free-form documents (~ SQL table row)

Data modeling

- Creating a data model with MongoDB does not have to follow the rules that apply for relational databases. Often, they should not.
- Consider these questions: is this a **composition** relationship (**containment**)? Is this "aggregate" of documents often used at the same time (i.e. can we reduce chattiness)? Would embedding lead to "a lot" of data **duplication**?



Normalized data model
(references)



Embedded data model
(sub-documents)

Install & start MongoDB



<https://www.mongodb.org>

After downloading MongoDB, you must follow the installation procedure for your platform:

<https://docs.mongodb.org/manual/installation/#tutorials>

Test MongoDB on Linux/OS X

As long as you have set up your **PATH** correctly, you should be able to enter the MongoDB console from your command line, and insert and find some data:

```
$> mongo
MongoDB shell version: 3.2.3
connecting to: test
> db.things.insert({ "fruit": "apple" })
WriteResult({ "nInserted" : 1 })
> db.things.insert({ "name": "John Doe", "age": 24 })
WriteResult({ "nInserted" : 1 })
> db.things.find()
{ "_id" : ObjectId("56ca09b5d536b4526d219ba8"), "fruit" : "apple" }
{ "_id" : ObjectId("56ca095ed536b4526d219ba7"), "name" : "John Doe", "age" : 24 }
```

Test MongoDB on Windows

MongoDB looks for a `/data/db` directory by default. You can create it in Babun with the following command:

```
$> mkdir -p /c/data/db
```

To run the MongoDB **server**, run the following command in Babun (adapt the path if your MongoDB installation is elsewhere):

```
$> "/c/Program Files/MongoDB/Server/3.2/bin/mongod.exe"  
  
2016-02-22T07:25:52.998+0100 I CONTROL  [initandlisten] MongoDB starting : pid=203 port=27017 dbpath=/opt/local/var/db/  
mongodb 64-bit host=Phaeton.local  
2016-02-22T07:25:52.999+0100 I CONTROL  [initandlisten] db version v3.0.7  
2016-02-22T07:25:52.999+0100 I CONTROL  [initandlisten] git version: nogitversion  
2016-02-22T07:25:52.999+0100 I CONTROL  [initandlisten] build info: Darwin tenten-slave.macports.org 14.5.0 Darwin Kernel  
Version 14.5.0: Wed Jul 29 02:26:53 PDT 2015; root:xnu-2782.40.9~1/RELEASE_X86_64 x86_64 BOOST_LIB_VERSION=1_49  
2016-02-22T07:25:52.999+0100 I CONTROL  [initandlisten] allocator: system  
2016-02-22T07:25:52.999+0100 I CONTROL  [initandlisten] options: { storage: { dbPath: "/opt/local/var/db/mongodb" },  
systemLog: { destination: "file", logAppend: true, path: "/opt/local/var/log/mongodb/mongodb.log" } }  
2016-02-22T07:25:57.514+0100 I NETWORK  [initandlisten] waiting for connections on port 27017
```

If you see **Waiting for connections on port 27017**, it means your MongoDB server is running. It will keep running until you hit Ctrl-C to stop it.

Test MongoDB on Windows

To run a MongoDB **client**, you must use the **Windows command line** and not Babun, as there are interaction bugs with Babun:

```
$> "C:\Program Files\MongoDB\Server\3.2\bin\mongo.exe"
MongoDB shell version: 3.2.3
connecting to: test
> db.things.insert({ "fruit": "apple" })
WriteResult({ "nInserted" : 1 })
> db.things.insert({ "name": "John Doe", "age": 24 })
WriteResult({ "nInserted" : 1 })
> db.things.find()
{ "_id" : ObjectId("56ca09b5d536b4526d219ba8"), "fruit" : "apple" }
{ "_id" : ObjectId("56ca095ed536b4526d219ba7"), "name" : "John Doe", "age" : 24 }
```

Getting started with MongoDB

We will learn more **tomorrow.**

This guide will instruct you on how to insert,
find, update and remove documents:

<https://docs.mongodb.org/getting-started/shell/>

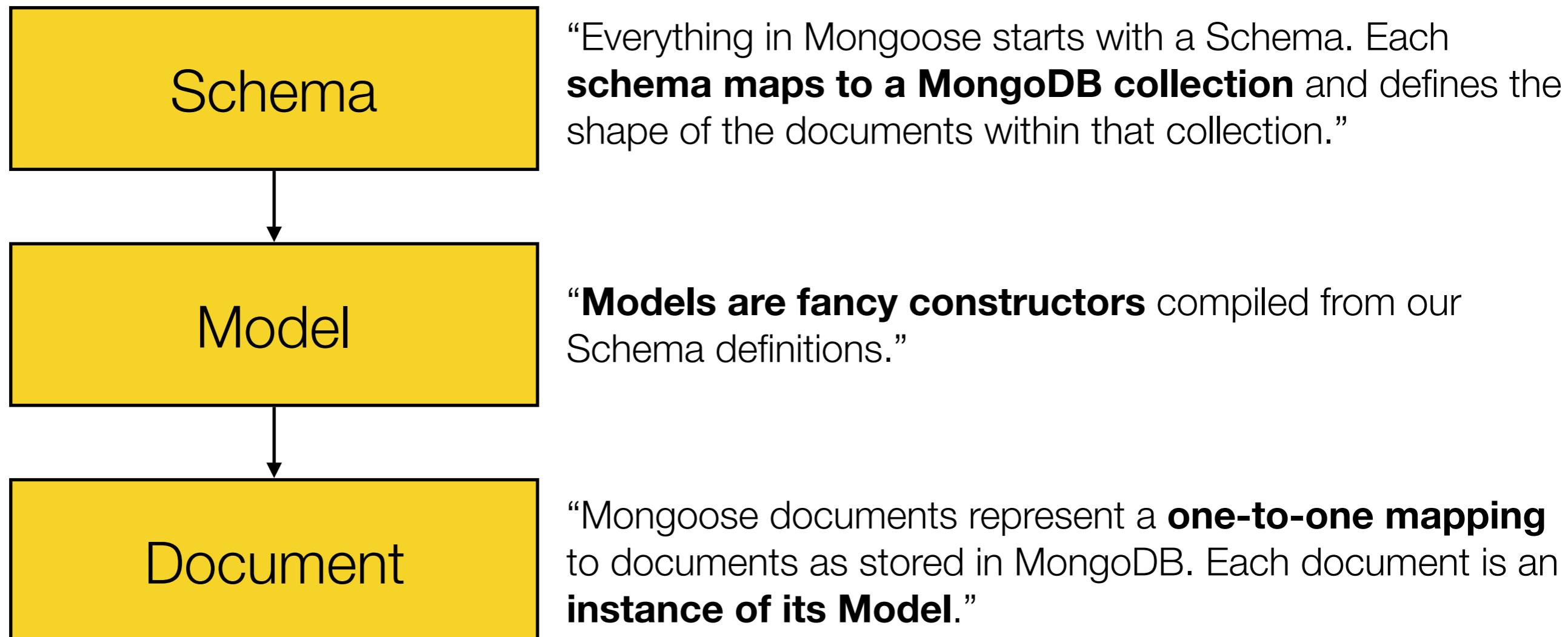
mongoose

elegant **mongodb** object modeling for **node.js**

Mongoose

Mongoose: an ODM for MongoDB

*“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.”*



Mongoose example

```
var mongoose = require('mongoose'),  
    Schema = mongoose.Schema;  
  
var PersonSchema = new Schema({  
  name: {  
    first: String,  
    last: { type: String, required: true }  
  },  
  age: { type: Number, min: 0 }  
});  
  
mongoose.model('Person', PersonSchema);
```

Define a **schema**

Create a **model** to manage documents with that schema and store them in a MongoDB **collection**

Create a **document** with the model

Save it →

```
var mongoose = require('mongoose'),  
    Person = mongoose.model('Person');  
  
var johnDoe = new Person({  
  name: {  
    first: "John",  
    last: "Doe"  
  },  
  age: 24  
})  
  
johnDoe.save(function(err) {  
  if (err) {  
    console.warn("John Doe could not be saved");  
  }  
});
```

Mongoose validations

Mongoose has **built-in validations**, and you can also **write your own**

```
var PersonSchema = new Schema({
  name: {
    first: String,
    last: { type: String, required: true }
  },
  age: { type: Number, min: 0 },
  phone: {
    type: String,
    validator: function(value) {
      return /^(\d{3})-(\d{2})-(\d{2})$/.test(value);
    },
    message: "{VALUE} must have the format 000-00-00"
  }
});
```

Ensure a property is present with **required**

Ensure a number is within bounds with **min/max**

Write your own **validator function** and **custom error message**

<http://mongoosejs.com/docs/validation.html>

Mongoose validations

If your document is invalid, the callback function will be called with an **error** describing which validations have failed.

```
johnDoe.save(function(err) {  
  if (err) {  
    console.warn("Validation error: " + err.message);  
  }  
});
```

<http://mongoosejs.com/docs/validation.html>

Mongoose queries

Person

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

you can chain conditions

retrieve only the fields you are interested in

Mongoose resources

Quick start guide

<http://mongoosejs.com/docs/index.html>

Queries

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

Full documentation

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

Grunt: The JavaScript Task Runner

What is Grunt?



Getting Started Configuring Tasks Plugins Documentation

GRUNT

The JavaScript Task Runner

Latest Version

- Stable: [v0.4.5](#) (npm)
- Development: [v1.0.0](#) (github)



[Discover Dev Tools](#), a free interactive course to help you master Chrome Dev Tools.

Ads by [Bocoup](#).

Why use a task runner?

In one word: automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes. After you've configured it through a [Gruntfile](#), a task runner can do most of that mundane work for you—and your team—with basically zero effort.

Why use Grunt?

The Grunt ecosystem is huge and it's growing every day. With literally hundreds of plugins to choose from, you can use Grunt to automate just about anything with a minimum of effort. If someone hasn't already built what you need, authoring and publishing your own Grunt plugin to npm is a breeze. See how to [get started](#).

Grunt plugins

```
{  
  "name": "webs-2016-example",  
  "version": "0.0.1",  
  "private": true,  
  "scripts": {  
    "start": "node app.js"  
  },  
  "dependencies": {  
    "express": "^4.13.3",  
    "serve-favicon": "^2.3.0",  
    "morgan": "^1.6.1",  
    "cookie-parser": "^1.3.3",  
    "body-parser": "^1.13.3",  
    "compression": "^1.5.2",  
    "method-override": "^2.3.0",  
    "glob": "^6.0.4",  
    "mongoose": "^4.1.2",  
    "jade": "^1.11.0"  
  },  
  "devDependencies": {  
    "grunt": "^0.4.5",  
    "grunt-develop": "^0.4.0",  
    "grunt-contrib-stylus": "^1.0.0",  
    "grunt-contrib-watch": "^0.6.1",  
    "request": "^2.60.0",  
    "time-grunt": "^1.2.1",  
    "load-grunt-tasks": "^3.2.0"  
  }  
}
```

The Yeoman express generator has already integrated Grunt and included a few **plugins** for us.

The Gruntfile

The Gruntfile is where Grunt plugins are **configured**.

```
module.exports = function (grunt) {  
  
  grunt.initConfig({  
    pkg: grunt.file.readJSON('package.json'),  
    develop: { ←  
      server: {  
        file: 'app.js'  
      }  
    },  
    stylus: { ←  
      dist: {  
        files: {  
          'public/css/style.css': 'public/css/style.styl'  
        }  
      }  
    },  
    ...  
  });  
};
```

The **grunt-develop** plugin runs our app and reloads it every time we make a change.

The **grunt-contrib-stylus** plugin compiles Stylus files to CSS.

And more...

You can run these plugins from the command line:

```
$> grunt develop  
$> grunt stylus
```

Composing tasks

You can run multiple plugins at once by registering a **task**.

```
grunt.registerTask('dev', [  
  'stylus',  
  'develop',  
  'watch'  
]);
```

\$> grunt dev

The **default** task is the task that is invoked if you simply run grunt without an argument.

```
grunt.registerTask('default', [  
  'stylus',  
  'develop',  
  'watch'  
]);
```

\$> grunt

RESTful API

GET PUT POST DELETE

Let's make a REST API

Testing tools



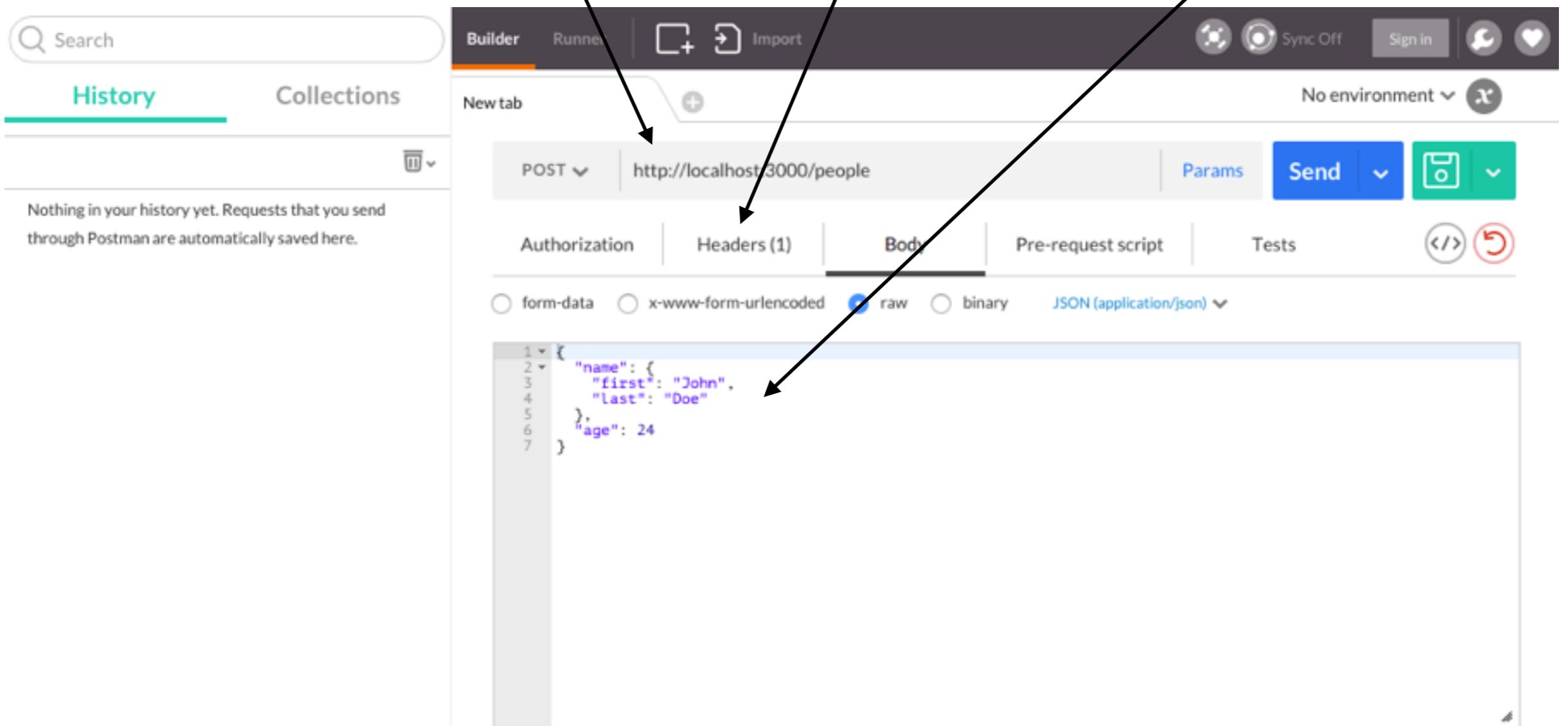
Google Chrome



Postman

Postman

URL HTTP headers Request body



A large, bold, blue logo consisting of the letters C, R, U, U, D arranged horizontally. The letters are thick and have a slight shadow, giving them a three-dimensional appearance.

Create

Read

Update Delete

CREATE RUD

Request

```
POST /people HTTP/1.1
Content-type: application/json

{
  "name": {
    "first": "John",
    "last": "Doe"
  },
  "age": 24
}
```

Response

```
HTTP/1.1 201 Created
Content-type: application/json

{
  "_id": "3orv8nrg",
  "name": {
    "first": "John",
    "last": "Doe"
  },
  "age": 24
}
```

The **POST** method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request URI.

HTTP 201 Created: The request has been fulfilled and resulted in a new resource being created.

Request

```
GET /people HTTP/1.1
```

Response

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

```
[  
  { "_id": "3orv8nrg", ... },  
  { "_id": "a08un2fj", ... }  
]
```

The **GET** method means retrieve whatever information (in the form of an entity) is identified by the Request URI.

HTTP 200 OK: Standard response for successful HTTP requests. In a GET request, the response will contain an entity corresponding to the requested resource.

Request

```
GET /people/3orv8nrg HTTP/1.1
```

Response

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

{
  "_id": "3orv8nrg",
  "name": {
    "first": "John",
    "last": "Doe"
  },
  "age": 24
}
```

Request

```
PUT /people/3orv8nrg HTTP/1.1
Content-type: application/json

{
  "name": {
    "first": "John",
    "last": "Smith"
  },
  "age": 34
}
```

Response

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

{
  "_id": "3orv8nrg",
  "name": {
    "first": "John",
    "last": "Smith"
  },
  "age": 34
}
```

The **PUT** method requests that the enclosed entity be stored under the supplied Request URI. If the Request URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server.

HTTP 200 OK: In a PUT request, the response will contain an entity describing or containing the result of the action.

CRU DELETE

Request

```
DELETE /people/3orv8nrg HTTP/1.1
```

Response

```
HTTP/1.1 204 No Content
Content-type: application/json
```

The **DELETE** method requests that the origin server delete the resource identified by the Request URI.

HTTP 204 No Content: The server successfully processed the request, but is not returning any content.

HTTP request methods

HTTP status codes

/d

ngénierie et de Gestion
(aud)



Technologies



YO



GRUNT



BOWER



mongoDB®

mongoose

express
web application
framework for
node



Postman

