MeetUp
MongoDB & Chirp

Antonio Di Motta
github.com/antdimot

Who am I?

I'm **Antonio Di Motta** e I'm Software Architect responsible for designing and developing of complex projects based on platform mixing open source and licensed products for the following markets: public transport, food and beverage, industry and media.

https://github.com/antdimot/chirp/blob/master/README.md

Chirp

The social engine open source, which has been developed using the MEAN javascript full stack.

An online demo is hosted on Azure. Read the release notes for the last updates. Activity board.

Features:

- public timeline
- user timeline
- user info
- post of a message
- list of the followers
- list of the following
- follow an user
- unfollow an user
- sign up
- log on

MEAN, the fullstack javascript

M stands for **MongoDB**, the world's leading **NoSQL** database. That's a kind of document type database that stores its data into a JSON-like formatted binary file called BSON (Binary JSON).

E stands for **Express**, a lightweight, minimalist framework built for Node.js. It's been created for web applications and APIs.

A stands for **AngularJS**, it's a client-side framework for MVC/MVVM done in JavaScript. The standard adopted is the closest to the MVVM pattern and is very robust and highly suitable for SPA.

N stands for **Node.js**, the foundation of Express. It runs on Chrome's V8 engine and is capable of non-blocking, event-driven I/O.

MongoDB – Why?

Today's solutions need to accommodate **tomorrow's needs**

- End of "Requirements Complete"
- Ability to economically scale
- Shorter solutions lifecycles

MongoDB

RDBMS		MongoDB
Database	→	Database
Table	→	Collection
Index	→	Index
Row	→	Document
Column	→	Field
Join	→	Embedding & Linking

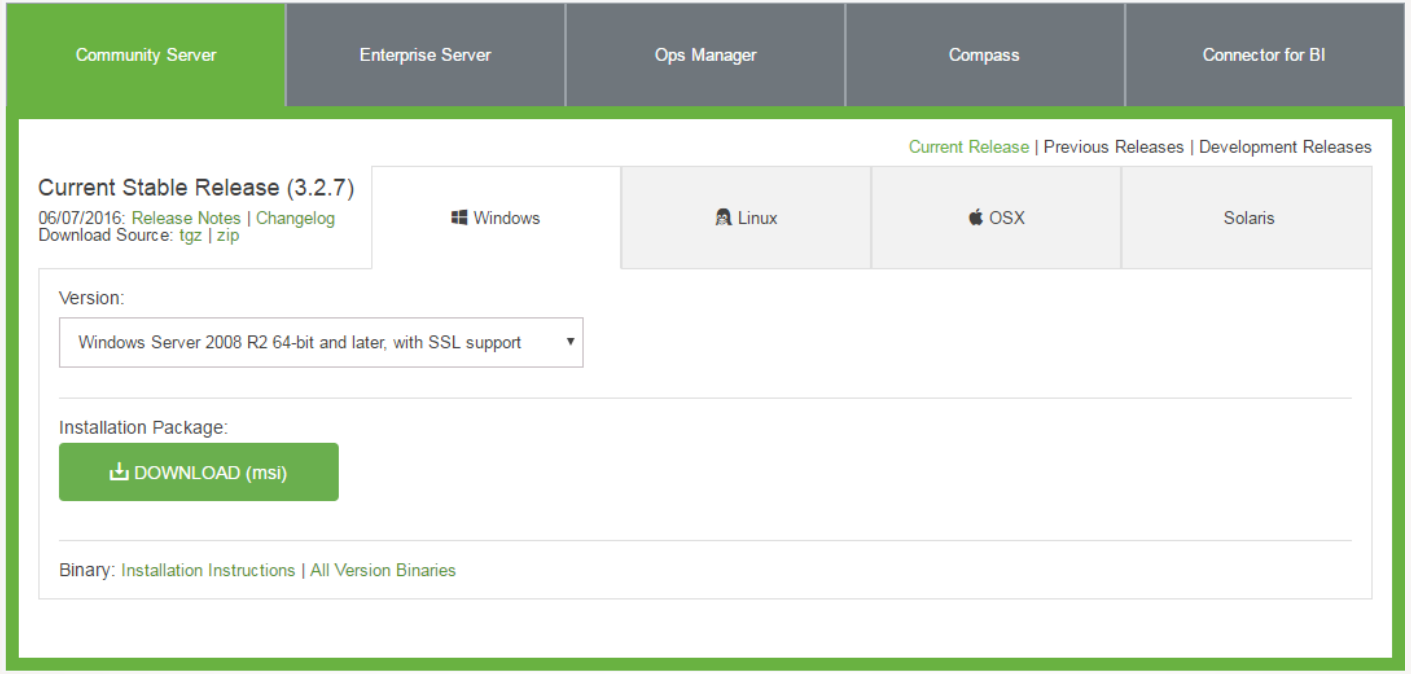
MongoDB – What is a document ?

```
// chirp, user document example
{
    "_id": "5759416fc7d0ffbdd72a2e95", // ObjectId
    "username": "dimotta",
    "displayname": "Antonio Di Motta",
    "password": "$2a$10$nn4S7KMtT8GzQhNBLnToJuBs",
    "email": "antonio.dimotta@gmail.com",
    "image": "dimotta.jpg",
    "following":["5759416fc7d0ffbdd72a2e96","5759416fc7d0ffbdd72a2e97"],
    "followers":["5759416fc7d0ffbdd72a2e96","5759416fc7d0ffbdd72a2e97"]
}
```

```
// chirp, post document example
{ "_ id": "5759416fc7d0ffbdd72a2e98",
    "username": "dimotta",
    "ownerid": "5759416fc7d0ffbdd72a2e95",
    "displayname": "Antonio Di Motta",
    "image": "dimotta.jpg ", "timestamp":
ISODate("2016-06-18")
    " text": "My first post on Chirp."
}
```

MongoDB – Installing

1) Downloading setup package from https://www.mongodb.com



2) Using DOCKER

docker pull mongo
docker run –d mongo

MongoDB – CLI

```
$ mongo
> show dbs
chirp
local
> use chirp
> show collections
posts
users
> db.users.find()
  {
    "_id": "5759416fc7d0ffbdd72a2e95",
- - - - - - - - - - - - - - - - - - -
```

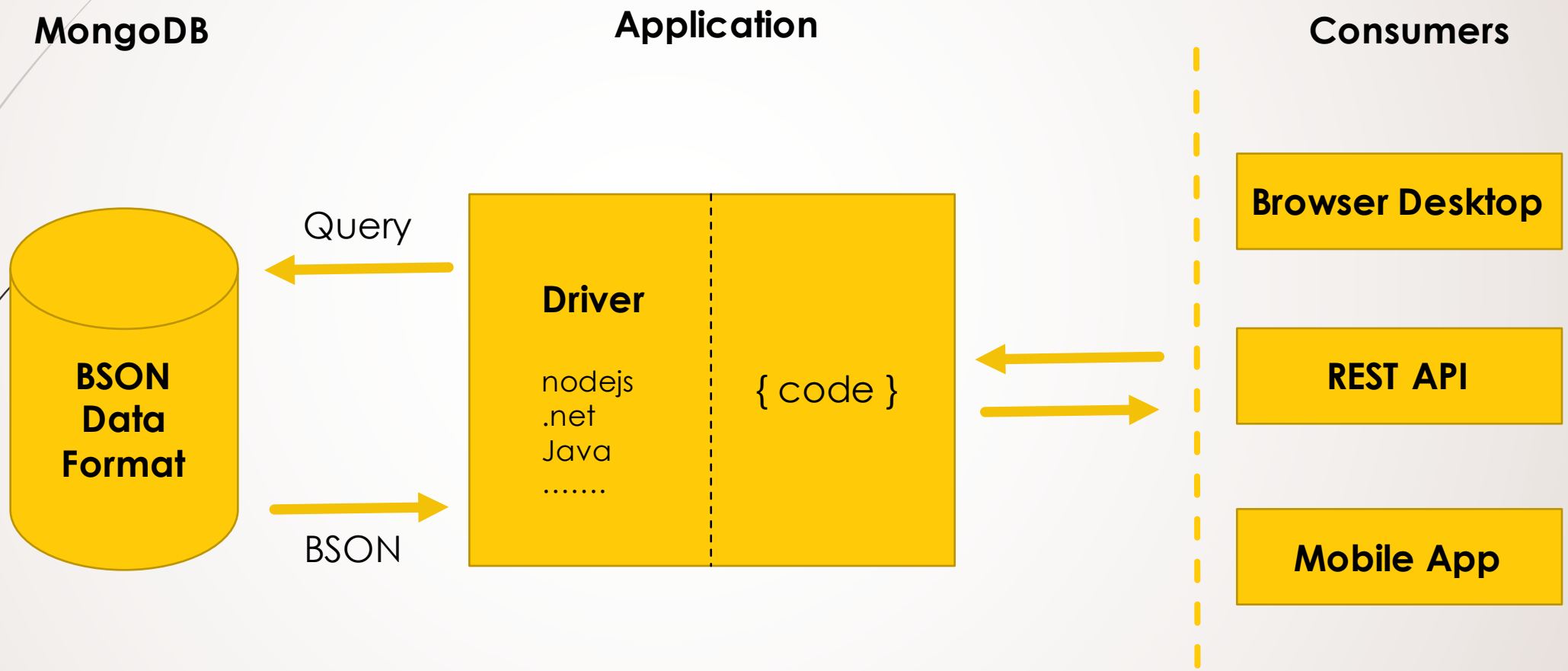
MongoDB – more with query

```
> myposts = db.posts.find(          // create a function
    { “username”: “dimotta” },      // only my posts
    { “text”:1, “timestamp”:1 },    // only text and timestamp fields
    )

> doSomethingWithPostList( myposts )
```

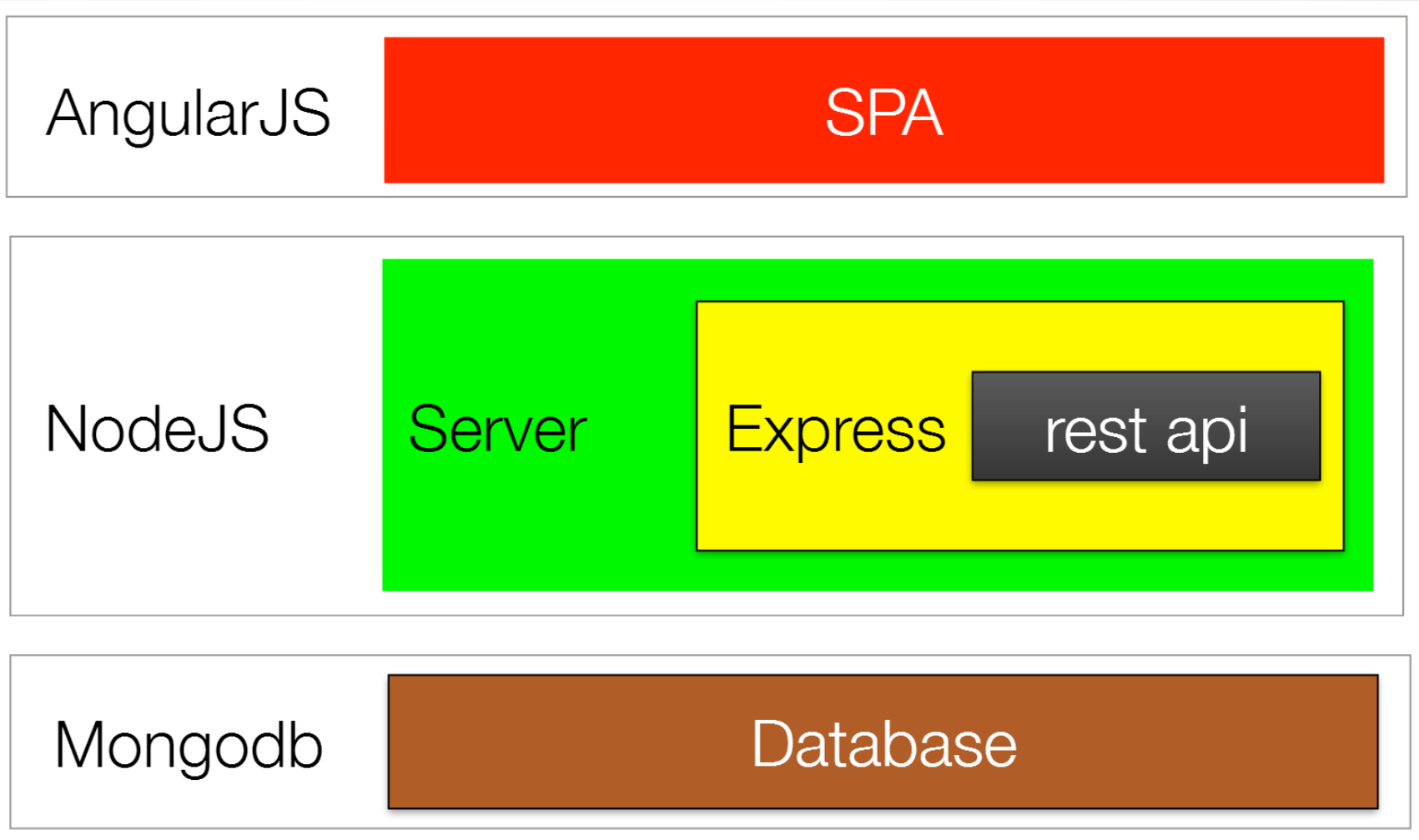
MongoDB – insert new document

```
> db.users.insert(
{
    "_id": "1",
    "username": "newusername",
    "displayname": "I'm not a bot :)",
    "password": password,
    "email": "email@",
    "image": "default.gif",
    "summary": "Only for testing :)",
    "following": [],
    "followers": []
});
```

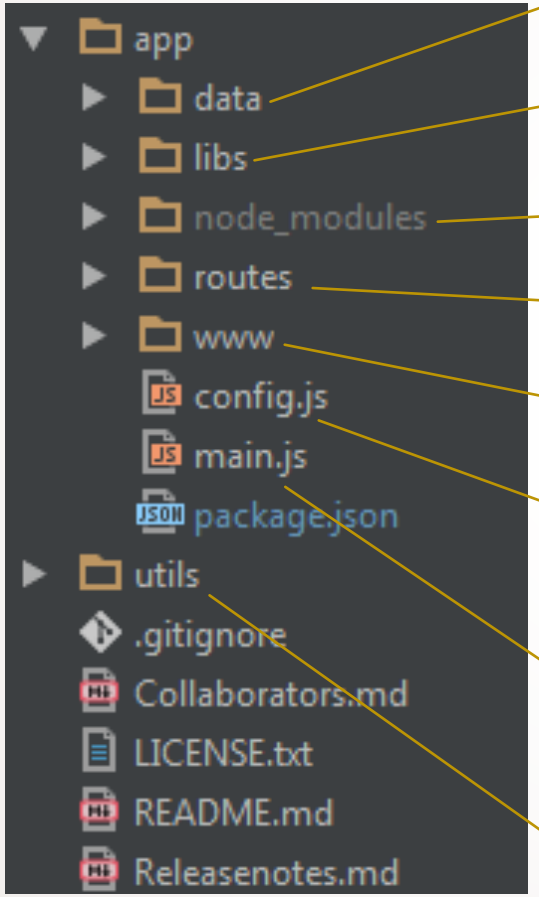
MongoDB – How to use it with app



Chirp architecture



How is organized the project



Static content files

Chirp utility library

External packages

Rest API

Front-end

Back-end configuration

Application entry point

Misc script files

app/package.json

```
{
  "name": "chirp",
  "homepage": "https://github.com/antdimot/chirp",
  "version": "0.2.6",
  "description": "The social engine open source",
  "license": "MIT",
  "main": "main.js",
  "scripts": {
    "start": "node main.js"
  },
  "author": "Antonio Di Motta",
  "dependencies": {
    "bcrypt": "^0.8.6",
    "body-parser": "^1.15.1",
    "cors": "^2.7.1",
    "express": "^4.13.4",
    "jsonwebtoken": "^5.7.0",
    "kerberos": "0.0.17",
    "mongodb": "^2.1.21",
    "morgan": "^1.7.0",
    "multer": "^1.1.0",
    "socket.io": "^1.4.6",
    "winston": "^2.2.0"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/antdimot/chirp.git"
  },
  "engines": {
    "node": ">=4.2.0",
    "npm": ">=2.14.0"
  }
}
```

app/libs/logger.js

```
module.exports = ()=> {
    var winston = require('winston');
    winston.emitErrs = true;

    var logger = new winston.Logger({
        transports: [
            new winston.transports.Console({
                level: 'debug',
                handleExceptions: false,
                json: false,
                colorize: true,
                timestamp: true
            })
        ],
        exitOnError: false
    });

    logger.stream = {
        write: (message, encoding)=> {
            logger.info(message);
        }
    };

    return logger;
};
```

app/libs/helper.js

```
module.exports = (logger)=> {
    return {
        action: {
            jsonResult: (req,res,data)=> {
                res.status(200).jsonp(data);
            },
            forbiddenResult: (req,res)=> {
                logger.warn('Forbidden at [%s]',req.url);
                res.status(403).jsonp({ message: '403 - Forbidden' });
            },
            notfoundResult: (req,res)=> {
                logger.warn('Error 404 at [%s]',req.url);
                res.status(404).jsonp({ message: '404 - Not found' });
            },
            errorResult: (err,req,res)=> {
                logger.error('Error 500 at [%s] details [%s]',req.url,err);
                res.status(500).jsonp({ message: '500 - Server error' });
            },
            okResult: (req,res)=> {
                res.sendStatus(200);
            }
        },
        string: {...}
    };
};
```

app/config.js

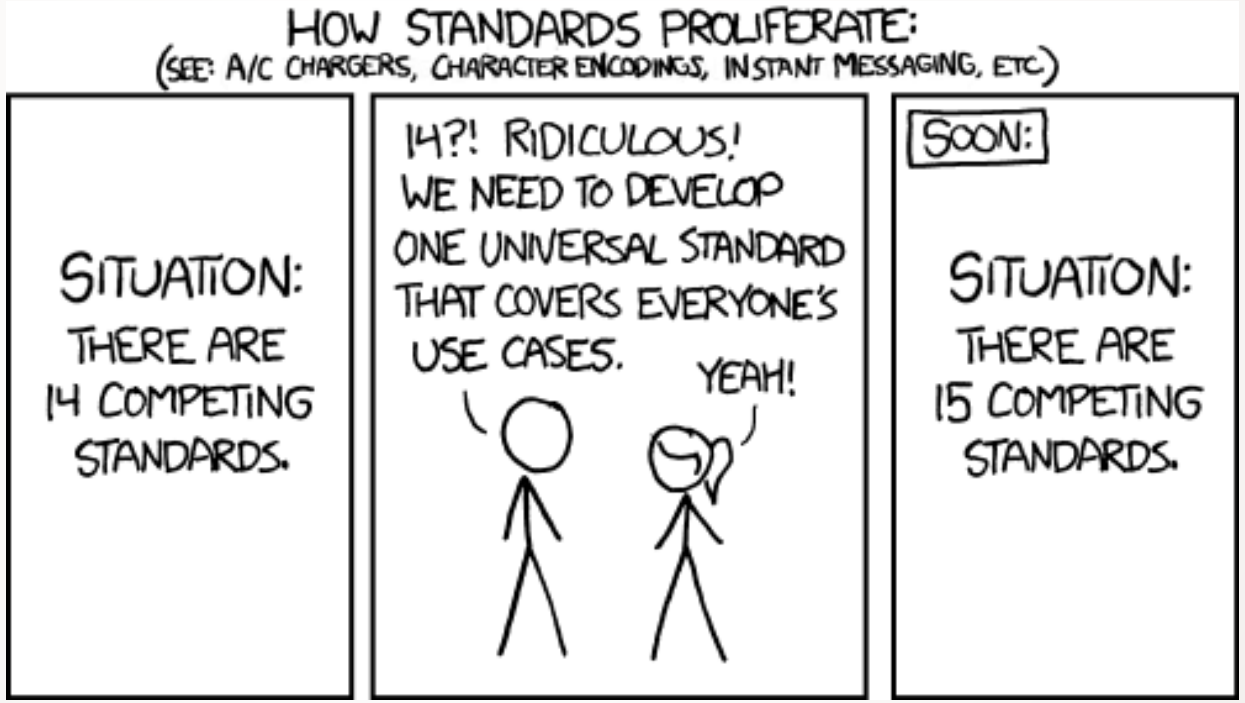
```
module.exports =
{
    server: {
        api: '/api/v1',
        limit: 20,
        imagepath: 'data/images'
    },
    image: 'default.png',
    mongodb: {
        connectionString: process.env.MDB || 'mongodb://localhost/chirp?autoReconnect=true'
    }
};
```

app/main.js

```
const express = require('express');
const app = express();
const http = require('http').Server(app);
const io = require('socket.io')(http);
const fs = require('fs');

const config = require('./config');
const logger = require('./libs/logger')();
const helper = require('./libs/helper')(logger);

const mongoClient = require('mongodb').MongoClient;
mongoClient.connect(config.mongodb.connectionString,
    (err, db) => {
        if(err) {...}

        const appContext = {     // create the application context
            config: config,
            helper: helper,
            logger: logger,
            db : db
        };

        app.use('/', express.static( __dirname + '/www'));
        app.use(require('./routes')(appContext));
        io.on('connection',(socket)=>
        {
            logger.debug('[Socket IO] client connected');
            socket.on('disconnect',()=>{...});
            socket.on('postmessage',(data)=>{...});
        });

        app.use( helper.action.notfoundResult );
        app.use( helper.action.errorResult );

        var port = process.env.PORT || 3000;
        http.listen(port,()=> {...});
});
```

APIs made simple

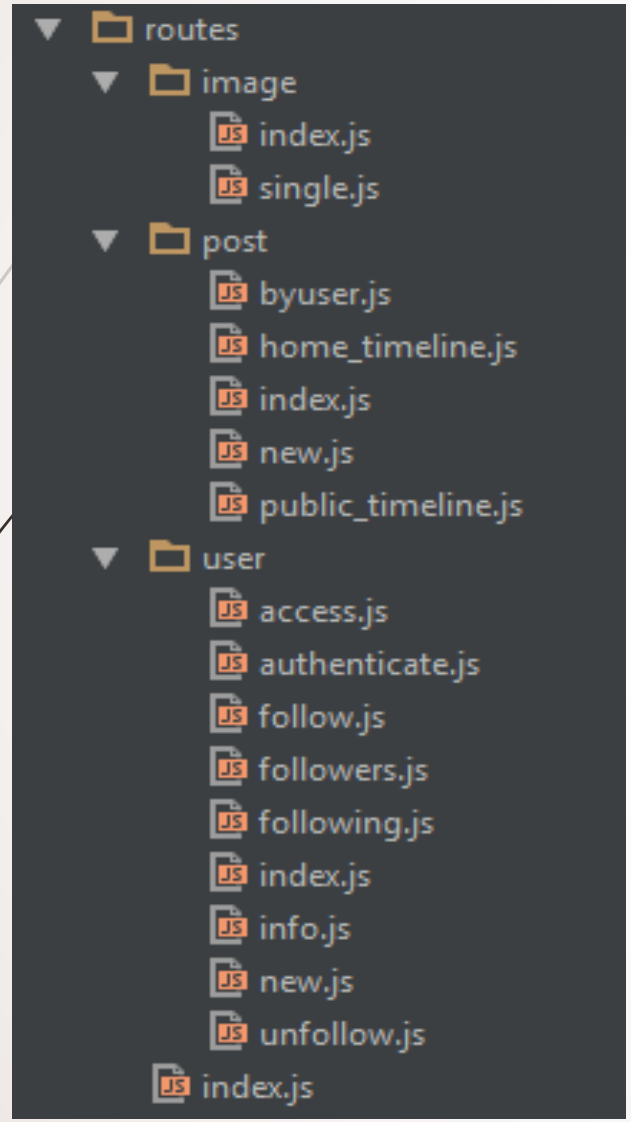
Basically each developer has his own standard and his own ideas on what an API should look like. **And that's bad.**



API best practices

- **Make it REST, keep it JSON**. Everybody speaks JSON, no need to ruin lives with XML
- **Never break backwards compatibility.** Version your API, nice and easy: ***/api/v1/user*** can easily coexist with **/api/v2/user** and your customer has the freedom to update his API when he is comfortable.
- **Never camelCase** it
- Never start a property name with a number
- **Pluralize arrays in naming**
- Booleans. Always true or false, **never null or undefined**
- **If a property has the value null then remove it**
- **Send your dates in UTC**, without any offsets. *2015–05–28T14:07:17Z* is much friendlier than *2015–05–28T14:07:17+00:00*
- **Use lowercased**, hyphen separated words in your URL. ***/store-order/1***
- Query params, as the field names should be snake-cased **customer_name, order_id**
- Use the right status codes: **200 for success, 403 forbidden…**
- Return proper error messages, never return error stack
- If the client does not need the entire resource he should be available to filter for **only the information interesting to him**, in order to save bandwidth.

app/routes



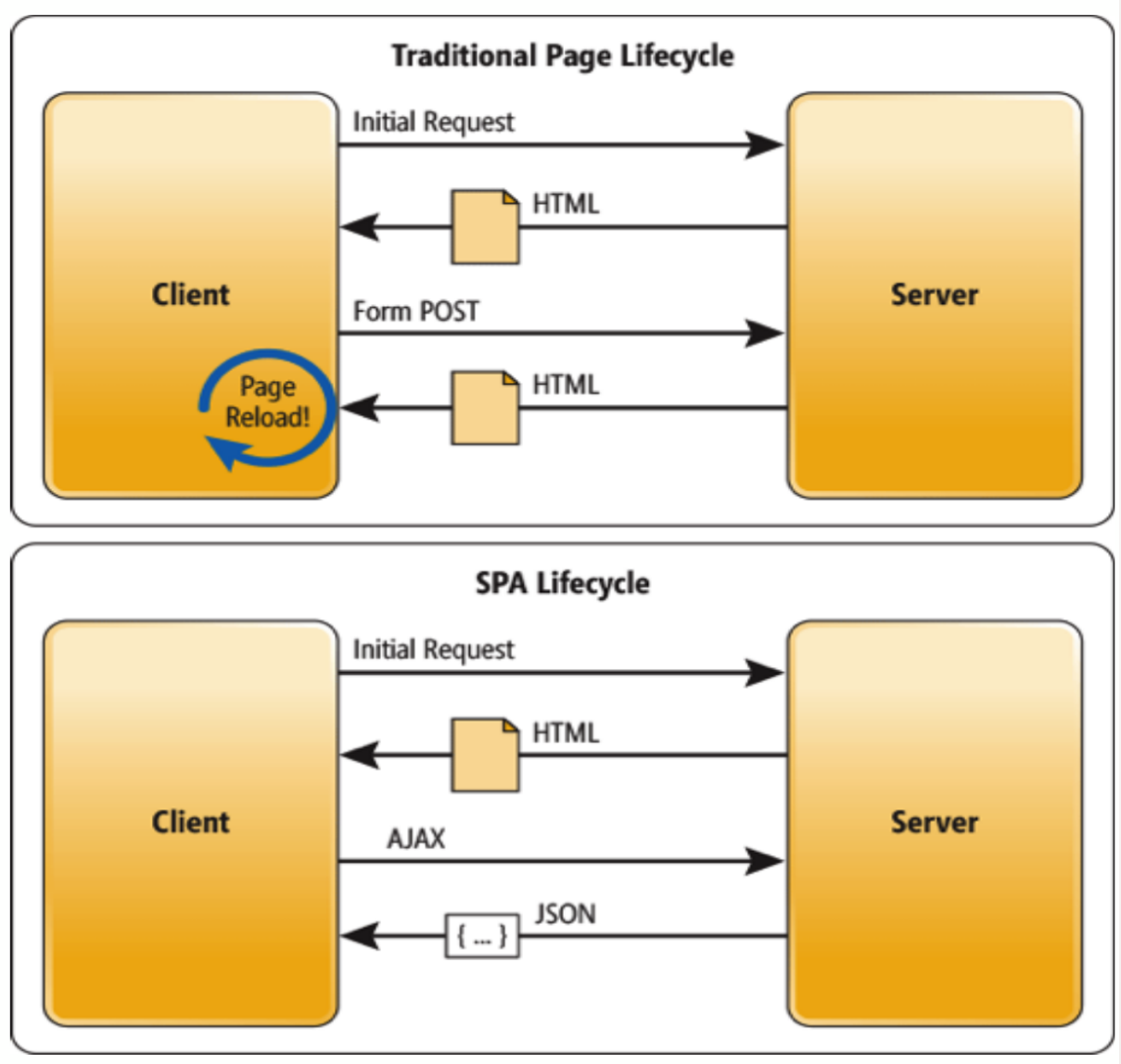
app/routes/index.js

```
module.exports = (ctx)=> {
    const routes = require('express').Router();

    routes.use(ctx.config.server.api + '/user', require('./user')(ctx));

    routes.use(ctx.config.server.api + '/post', require('./post')(ctx));

    routes.use(ctx.config.server.api + '/image', require('./image')(ctx));

    return routes;
}
```

app/routes/post/home_timeline.js

```
module.exports = (ctx) => {
    return (req,res) => {
        ctx.db.collection('users').findOne(
            {'username':req.params.username},
            {'fields':{'_id':1,'following':1,'displayname':1,'image':1} },
            (err,data) => {
                if (data) {
                    data.following.push(data._id); // add my id for showing also my posts

                    ctx.db.collection('posts')
                        .find(  {'ownerid':{$in:data.following} }, {'limit':ctx.config.server.limit,'sort':{'timestamp':-1}})
                        .toArray((err,items)=> {
                            if(err) return ctx.helper.action.errorResult(err.message,req,res);

                            items.forEach((element)=> {
                                element.text = ctx.helper.string.bodyProcess(element.text); // process the body
                                element.imagepath = ctx.config.server.api + '/image/' + element.image; // added image resource api
                            });

                            ctx.helper.action.jsonResult(req,res,items);
                        });
                }
                else {
                    ctx.helper.action.okResult(req, res);
                }
        });
    };
}
```

Single Page lifecycle model



AngularJS is a full-featured SPA framework

It supports both types of binding one-way and two-way data bindings

It supports MVC pattern

It supports static template and angular templates

You can add custom directives

It supports REST full services

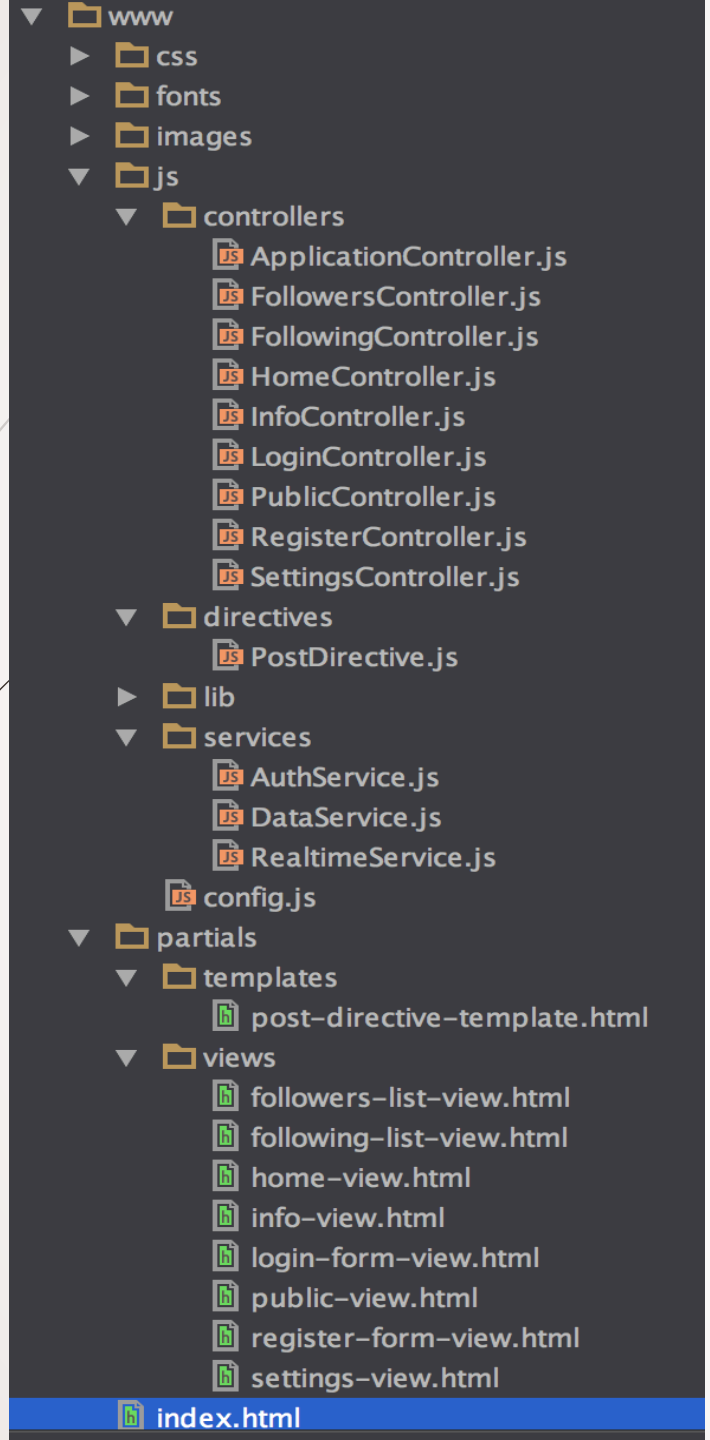
It supports form validations

It supports both client and server communication

It supports dependency injection

Event Handlers

Remember WWW folder?



app/www/index.html (only body)

```
<div class="container-fluid" ng-controller="AppController as appCtrl">
    <div class="row">
        <div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">
            <nav class="navbar navbar-default navbar-fixed-top">
                <div class="container">
                    <div class="navbar-header">
                      <a class="navbar-brand" href="#/public">Chirp</a>
                    </div>
                    <p class="navbar-text navbar-right">
                        <span ng-show="appCtrl.islogged">
                            <a href="#/home">{{appCtrl.user.displayname}}</a>
                        </span>  
                        <a class="btn btn-info btn-xs" href="#/signup" role="button" ng-show="!appCtrl.islogged">sign up</a>
                        <a class="btn btn-default btn-xs" href="#/login" role="button" ng-show="!appCtrl.islogged">login</a>
                        <a class="btn btn-default btn-xs" ng-click="appCtrl.logout()" role="button" ng-show="appCtrl.islogged">logout</a>
                    </p>
                </div>
            </nav>
        </div>
    </div>
    <div class="row" style="...">
        <div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">

            <div ui-view></div>

        </div>
    </div>
</div>
```

app/www/js/config.js

```
(function(){
    'use strict';

angular.module('appChirp', ['ui.router','ngSanitize','ngCookies'])
    .constant("config",
    {
      "api": "/api/v1"
    })
    .config(['$logProvider','$stateProvider','$urlRouterProvider', function($logProvider,$stateProvider,$urlRouterProvider)
    {
      $logProvider.debugEnabled(true);

      $urlRouterProvider.otherwise("/public");

      $stateProvider
        .state('public', {
            url: "/public",
            templateUrl: 'partials/views/public-view.html',
            controller: 'PublicController',
            controllerAs: 'vm'
        })
        .state('home', {
            url: "/home",
            templateUrl: 'partials/views/home-view.html',
            controller: 'HomeController',
            controllerAs: 'vm'
        })
        .state('login', {
            url: "/login",
            templateUrl: 'partials/views/login-form-view.html',
            controller: 'LoginController',
            controllerAs: 'vm'
        })
        .state('following', {
            url: "/following",
            templateUrl: 'partials/views/following-list-view.html',
            controller: 'FollowingController',
            controllerAs: 'vm'
        })
        .state('followers', {
            url: "/followers",
            templateUrl: 'partials/views/followers-list-view.html',
            controller: 'FollowersController',
            controllerAs: 'vm'
        })
```

app/www/js/services/dataservice.js

```
(function() {
    'use strict';

    angular.module('appChirp')
        .factory("DataService", ['$http', '$log', 'config',
        function ($http, $log, config)
        {
            return {
                getPublicPostList: function (callBack) {
                    $http.get(config.api + "/post/public")
                        .success(function (data) {
                            callBack(data);
                        })
                        .error(function(){
                            callBack();
                        });
                },
```

app/www/js/services/authservice.js

```
(function() {
    'use strict';

    angular.module('appChirp')
            .factory("AuthService", ['$http', '$log', 'config', 'DataService',
            function ($http, $log, config, DataService) {
              var _authUser = null;

             return {
                 login: function (credentials,callBack)
                 {
                     var username = credentials.username;
                     var password = credentials.password;

                     DataService.getUserByCredentials(username,password,
                         function(data) {
                             if(data)
                             {
                                 $log.debug("[%s] Logged in user: %s",
                                            new Date().toISOString(),
                                            data.username);

                                 _authUser = data;

                                 callBack(_authUser.username);
                             }
                             else callBack();
                         }
                     );
                 },
```

Socket.IO enables real-time bidirectional event-based communication.
It works on every platform, browser or device, focusing equally on reliability and speed.

```
(function() {
    'use strict';

    angular.module('appChirp')
        .factory("RealtimeService", ['$http', '$log', 'config',
        function ($http, $log, config)
        {
            var socket = io();

            return {
                postMessage: function (data) {
                  socket.emit('postmessage',data);
                },
                onMessage: function (callBack) {
                  socket.on('postmessage',function(data){
                    callBack(data);
                  });
                }
            };
        }]);
})();
```

RealtimeService.js

socket.io

```
(function() {
    'use strict';
    angular.module('appChirp').controller('HomeController',
        ['$scope','$log','$location','$timeout','$cookies','DataService','AuthService','RealtimeService','$sanitize',
        function ($scope,$log,$location,$timeout,$cookies,DataService,AuthService,RealtimeService,$sanitize) {
            var ctrl = this;

            ctrl.loadPosts = function () {
                DataService.getHomePostList(ctrl.user.username,
                    function (data) {
                        ctrl.posts = data;
                    });
            };

            RealtimeService.onMessage(function () {
                ctrl.loadPosts();
            });
        }]);
})();
```

HomeController.js

PostDirective.js

```
(function() {
    'use strict';

    angular.module('appChirp')
        .directive("post",['$log','DataService','RealtimeService','AuthService',
        function ($log,DataService,RealtimeService,AuthService)
        {
          return {
            restrict: 'E',
            templateUrl: 'partials/templates/post-directive-template.html',
            replace: false,
            scope: {
              data: '=post',
              logged: '='
            },
            controller: function () {
              var ctrl = this;
              ctrl.repost = function (postid) {
                  alert('This feature is not yet implemented!');
              };
            },
            controllerAs: 'vm'
          };
        }]);
})();
```

post-directive-template.html

```
<div class="panel panel-default">
  <div class="panel-body">
    <span style="...">
      <img class="img-rounded" ng-src="{{data.imagepath}}" width="55px" height="55px"> 
    </span>
    <span>
      <b><a href="#/info/{{data.username}}">{{data.displayname}}</a></b>
      <small>{{data.timestamp | date:'MM/dd/yyyy @ h:mma' }}</small>
      <div ng-bind-html="data.text"></div>
    </span>
  </div>
</div>
```

Chirp, to do:

- searching (users and messages)
- security enhancements (ie. add jwt)
- hashtag support
- api documentation
- edit user information
- customize user info (ie. image profile)
- repost

Happy coding ☺