ArangoDB Foxx

Creating APIs for Single Page Web Applications

by Lucas Dohmen

moonglum

moonbeamlabs

RWTH Aachen, Computer Science Student on branch master
triAGENS GmbH, Developer

Single Page Web Applications

The Idea

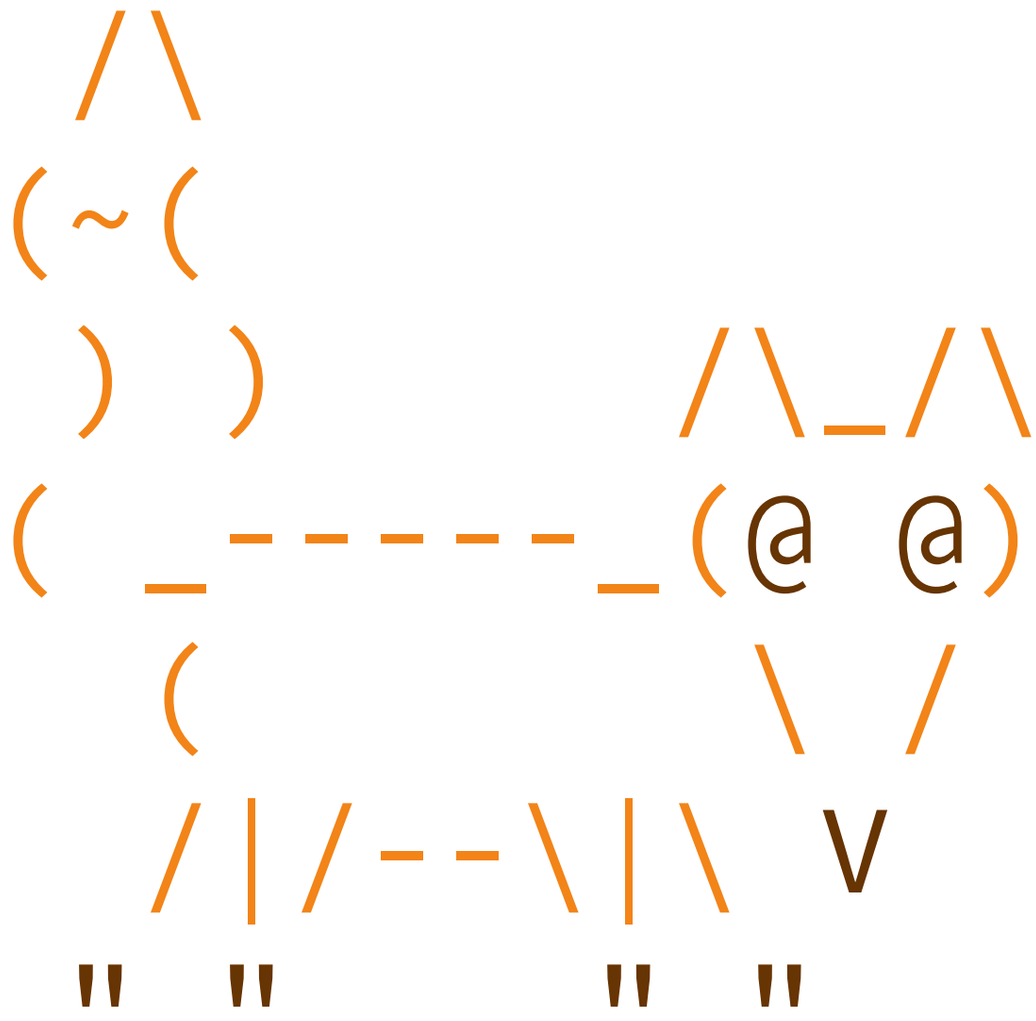
- What if we could talk to the database directly?
- It would only need an API
- What if we could define this API in JavaScript?

Single Page Web Applications

This doesn‘t mean its a Rails/… Killer

What is  ArangoDB ?

- Free and Open Source...
- ... Document and Graph Store...
- ... with embedded JavaScript...
- ... and an amazing query language



- An easy way to define REST APIs on top of ArangoDB
- Tools for developing your single page web application

Why another solution?

- ArangoDB Foxx is streamlined for API creation – not a Jack of all trades
- It is designed for front end developers: Use JavaScript, you already know that (without running into callback hell *cough* Node.js)

Foxx.Application

```
FoxxApplication = require("org/arangodb/foxx").Application;
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.get("/users", function(req, res) {
});
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.get("/users", function(req, res) {
  res.set("Content-Type", "text/plain");
});
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.get("/users", function(req, res) {
  res.set("Content-Type", "text/plain");
  res.body = "Worked!";
});
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.get("/users", function(req, res) {
  res.set("Content-Type", "text/plain");
  res.body = "Worked!";
});

app.start(applicationContext);
```

Parameterize the routes

- You may want a route like `users/:id`...
- ...and then access the value of `id` easily

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.get("/users", function(req, res) {
  res.set("Content-Type", "text/plain");
  res.body = "Worked!";
});

app.start(applicationContext);
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.get("/users/:id", function(req, res) {
  res.set("Content-Type", "text/plain");
  res.body = "Worked!";
});

app.start(applicationContext);
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.get("/users/:id", function(req, res) {
  res.set("Content-Type", "text/plain");
  res.body = "Worked: " + req.params("id");
});

app.start(applicationContext);
```

Manifest.json

- In your Foxx.Application you describe your routes
- But your application can consist of multiple Foxx.Applications
- … and you also want to deliver assets and files

```
{
  "name": "my_website",
  "version": "1.2.1",
  "description": "My Website with a blog and a shop",
  "thumbnail": "images/website-logo.png",

  "apps": {
    "/blog": "apps/blog.js",
    "/shop": "apps/shop.js"
  },

  "assets": {
    "application.js": {
      "files": [
        "vendor/jquery.js",
        "assets/javascripts/*"
      ]
    }
  }
}
```

```
{
  "name": "my_website",
  "version": "1.2.1",
  "description": "My Website with a blog and a shop",
  "thumbnail": "images/website-logo.png",

  "apps": {
    "/blog": "apps/blog.js",
    "/shop": "apps/shop.js"
  },

  "assets": {
    "application.js": {
      "files": [
        "vendor/jquery.js",
        "assets/javascripts/*"
      ]
    }
  }
}
```

```
{
  "name": "my_website",
  "version": "1.2.1",
  "description": "My Website with a blog and a shop",
  "thumbnail": "images/website-logo.png",

  "apps": {
    "/blog": "apps/blog.js",
    "/shop": "apps/shop.js"
  },

  "assets": {
    "application.js": {
      "files": [
        "vendor/jquery.js",
        "assets/javascripts/*"
      ]
    }
  }
}
```

More

- Define a **setup** and **teardown** function to create and delete collections
- Define **lib** to set a base path for your require statements
- Define **files** to deliver binary data unaltered

Documentation as a first class citizen

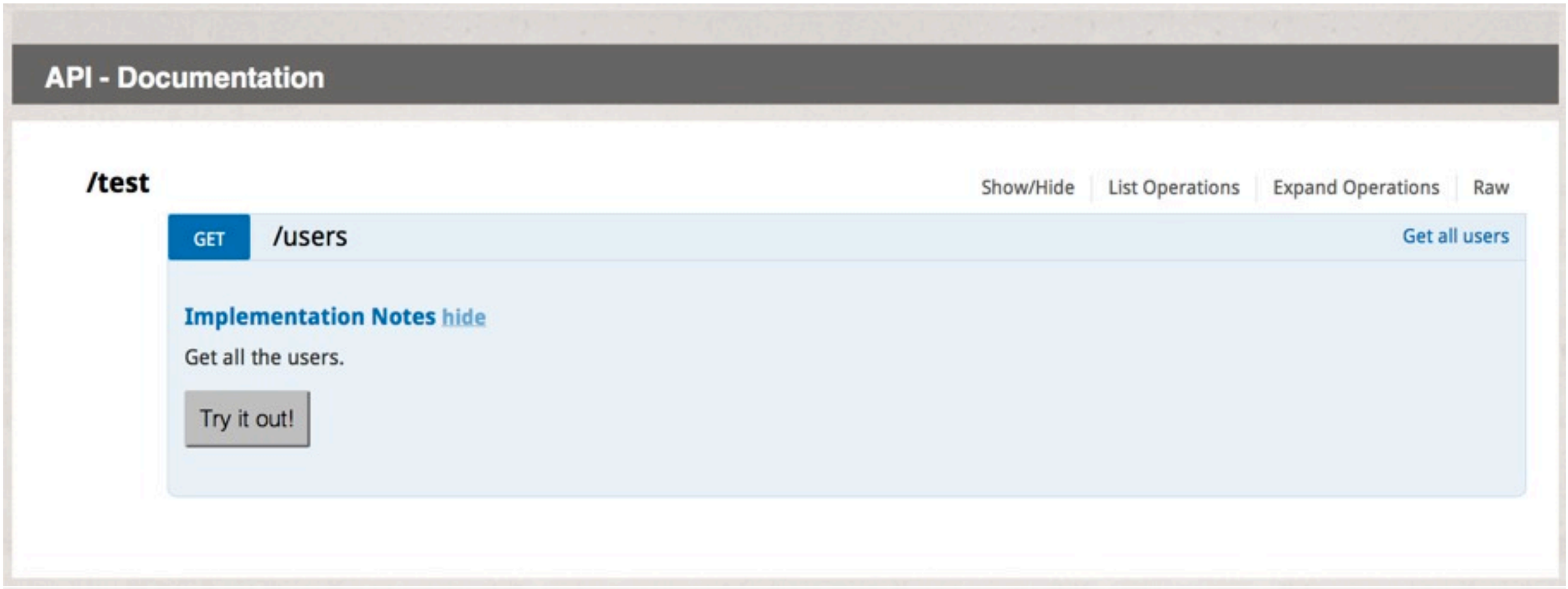
Annotate your Routes

- For Documentation
- But will later also be used for validation etc.

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.get("/users/:id", function(req, res) {
  res.set("Content-Type", "text/plain");
  res.body = "Your Wiese: " + req.params("id");
});

app.start(applicationContext);
```

```
app.get("/users/:id", function(req, res) {
  res.set("Content-Type", "text/plain");
  res.body = "Your User: " + req.params("id");
}).pathParam("id", {
    description: "ID of the User",
    dataType: "int"
});
```

Automatically generate Swagger Docs



Simple Demo

In the first part

- The motivation behind ArangoDB Foxx
- Foxx.Application and Manifests
- Documentation

Foxx:

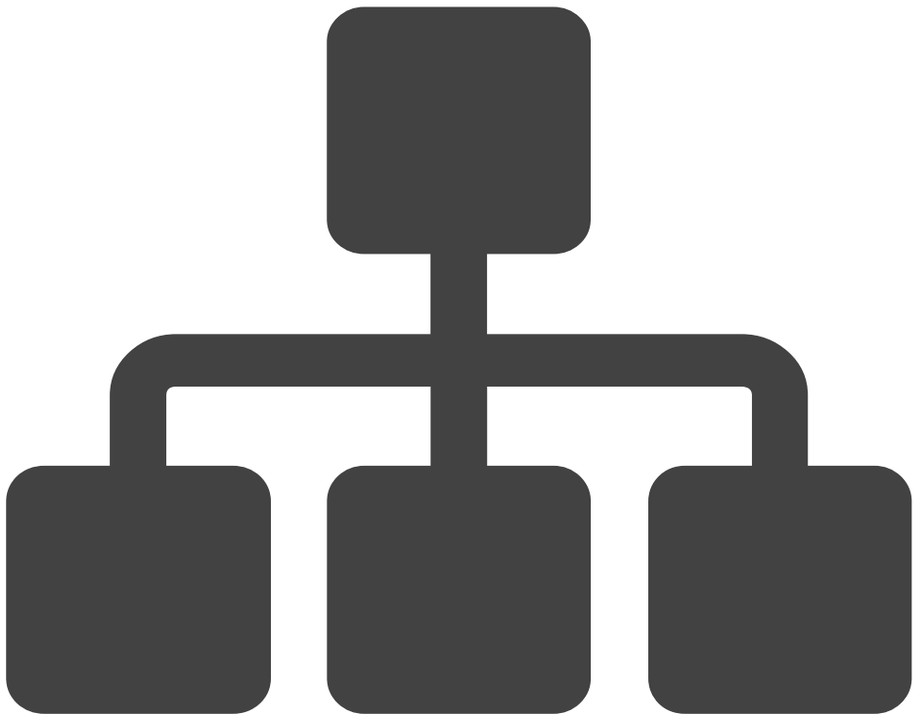
Repository and Model

“

Good design is about
drawing lines

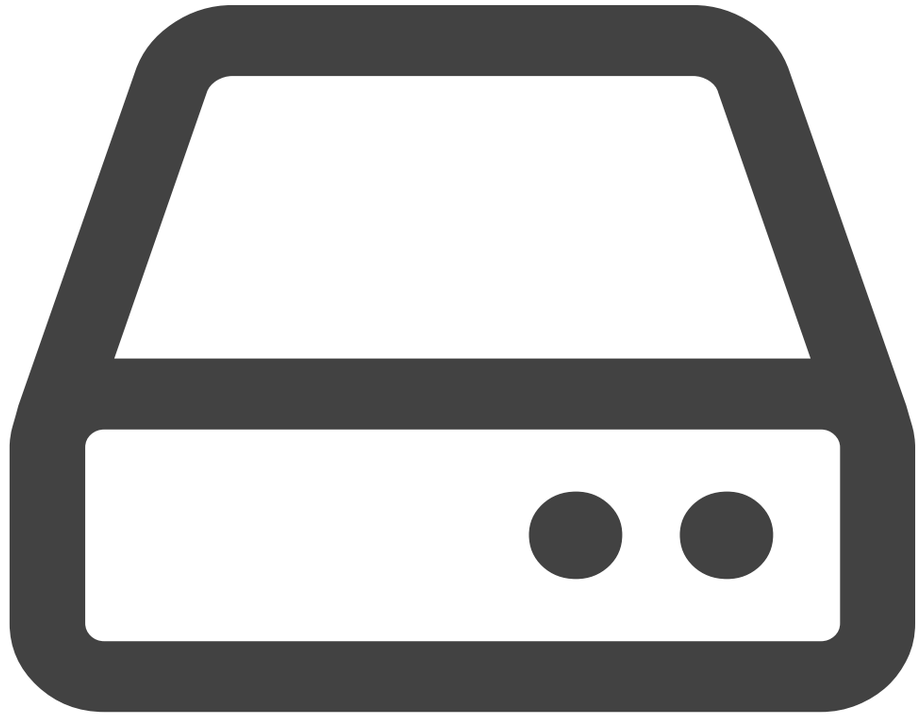
Uncle Bob in
Architecture the Lost Years (Ruby Midwest 2011)

Domain Models



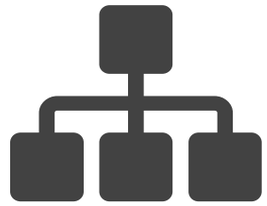
Foxx.Model

Persistence



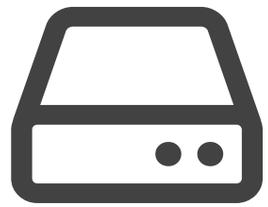
Foxx.Repository

Foxx.Model



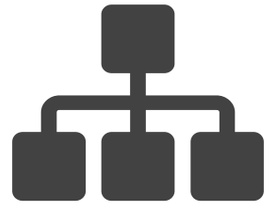
- Representation of the data
- Convenience Methods
- Validation

Foxx.Repository



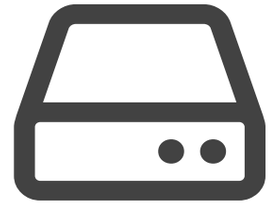
- Save and Retrieve Data
- Simple Queries
- Define your own queries

Foxx.Model



- The **constructor** takes a hash of attributes
- Access the attributes with **get**, **set** and **has**
- The **toJSON** will be used to write the data into the database

Foxx.Repository



- The **constructor** takes a **collection** and the **prototype of the model**
- **save** for example expects an instance of the model
- **firstExample** finds a suitable dataset and returns it as an instance of the model
- Other methods: **remove**, **replace**, **update**…

You need more?

- Use **Foxx.Repository.extend** and **Foxx.Model.extend** to inherit from the prototype
- Add your own methods
- Your extensions live in separate files

```
Foxx = require("org/arangodb/foxx");

MyRepository = Foxx.Repository.extend({
});

exports.Repository = MyRepository;
```

```
Foxx = require("org/arangodb/foxx");

MyRepository = Foxx.Repository.extend({
  byName: function (name) {
    return this.byExample({
      name: name
    });
  }
});

exports.Repository = MyRepository;
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.get("/foxx/:name", function (req, res) {
});
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.registerRepository("fancyCollection", {
  model: "models/my_model",
  repository: "repositories/my_repository"
});

app.get("/foxx/:name", function (req, res) {
});
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.registerRepository("fancyCollection", {
  model: "models/my_model",
  repository: "repositories/my_repository"
});

app.get("/foxx/:name", function (req, res) {
  var name = req.params("name"),
    item = repositories.fancyCollection.byName(name);
});
```

```
FoxxApplication = require("org/arangodb/foxx").Application;

app = new FoxxApplication();

app.registerRepository("fancyCollection", {
  model: "models/my_model",
  repository: "repositories/my_repository"
});

app.get("/foxx/:name", function (req, res) {
  var name = req.params("name"),
    item = repositories.fancyCollection.byName(name);
  res.json(item.toJSON());
});
```

Why this separation?

- It doesn‘t violate the SRP like ActiveRecord
- In a lot of cases you can use the standard Repository or Model and don‘t need your own
- It‘s great for testing
  - You can mock the collection and the model prototype to test your Repository
  - You don‘t need to mock anything to test your model

Foxx:

The Ecosystem

Aal, Aas and a bowl of fish

- Look at demo applications
- Install components from a central repository
- Share your ideas

Foxx Demo App: Aye-Aye

Foxx:

The Future

A Glance into the Future

- Authentication
- Configure filters for your assets:
    - Coffee, Sass, UglifyJS2, Client Templates…
- Models will generate JSON Schemata
- Later: Generators, Caching, Logging, Testing…

Thanks

- Please try ArangoDB Foxx
- We ❤ to get feedback

Contact

- lucas@arangodb.org
- @moonbeamlabs on Twitter

Thanks

- Database icon designed by Romeo Barreto from The Noun Project
- Browser icon designed by Fernando Vasconcelos from The Noun Project
- Logos from Node.js, Ruby on Rails, Django and Symfony from the respective projects
- All other icons are from Font Awesome