MaaP: MongoDB as an admin Platform

User Manual

1 Introduction

1.1 Document purpose

This document wants to be a reference guide for a developer user.
This will guide you through the installation process, explaining the various options and commands that are available, and through the configuration and maintenance of the MaaP framework. This guide is fairly technical and intended for a reader who's familiar with the javascript sintax and noSQL databases. It is not to be intended as an end user guide.

2 MaaP

Hello and welcome to MaaP User Manual. In this manual we'll go through all the stuff you need to do to get MaaP up and running in no time, while also taking a look at some more advanced features.

3 Prerequirements

The only thing you need before installing MaaP is node.js. Node.js is used to run the server side of MaaP and the node package manager (npm) is also required to install al the dependecies of the framework. You can download the latest version of node.js from the official website

http://nodejs.org/

4 Quick Start

The quickest way to get MaaP is to get it directly from npm using

npm install -g maaperture

You can also install it from a local npm package using

npm install . -g

The "-g" option allows MaaP to be installed globally and thus allows you to launch it from everywhere on your computer.
We will go through all the other options in section 7.1.

After you've installed MaaP you want to create your first project.
To do this open a terminal in the directory you want your project to be created in and just type

maaperture create -N myFirstProject

This will create a new project named myFirstProject.
Now you just need to type

npm start

and your server will be up and running!

MaaP uses some default settings, but more than likely you will want to change them. In order to do so we have put all the configurable setting in a configuration file which will be described in depth in the next section.

Once you have configured MaaP as you need the last thing you have to do is to describe your pages and the data you want to display. In order to do so you will need to create some description files, one per every page you plan to make, using our DSL. This last step will be described in the section 6.

5 Configuration file

The configuration file is located inside maap_ project and by default is configured as so:

```
exports.app = app = {
        env: 'development',
        title: 'Maaperture',
        description: 'MongoDB as an Admin Platform - Aperture Software',
        host: 'localhost',
        port: 9000,
        ssl: false
}
```

This first snippet contains the title and description of your application, the address where your server will be hosted, the port used and the ssl enable or disable setting.

```
exports.adminConfig = {
        usersPerPage: 20,
        queriesPerPage: 2,
        queriesToShow: 100,
        indexesPerPage: 20
}
```

This second snippet contains setting about the web pages that the framework will generate such as the number of users profile displayed per page in the users collection, the number of queries per page on the queries collection, the total queries to show and the indexes per page on the indexes collection.

```
exports.session = {
        secret: 'boomShakalaka!YO',
        max_age: 3600000 // one hour (60s * 60m * 1000ms)
}
```

This snippet contains setting regarding the session, such as the session secret and the time of expiration of session cookies.

```
exports.userDB = {
        host: 'localhost',
        port: 27017,
        db_name: 'userdb',
        usersCollection: 'users',
        queryCollection: 'query'
}

exports.dataDB = {
        host: 'localhost',
        port: 27017,
        db_name: 'datadb'
}
```

These two snippets contains the settings of the user database and business database. You can specify the host of both databases, the port used, and the name.

For the user database you can also specify the collection containing the users data and the query manager.

```
exports.static_assets = {
        dir: __dirname + '/app',
        views: __dirname + '/views',
        dsl: __dirname + '/dsl'
}
```

This last snippets sets the directory used by the framework and should not be messed with unless by expert developers.

6 Domain Specific Language

The DSL is one of the most crucial part of MaaP.
Our DSL allows you to describe how you want your data to be displayed to the end user, so the better you can master it the more you'll get from our framework. You can define any number of columns, name them as you want, populate them directly from your database or by an elaborate query, sort them and much more.

We'll go through each and every available feature in the next paragraph and to do this we'll analyze an example of DSL file.

```
collection = {
        label: 'collection label',
        name: 'maapertureCollection',
        position: 3,

        index : {
                populate: [{collection: 'collection_name', key: 'document_key'}],
                perpage: 20,
                sortby: 'age',
                order: 'asc',
                column : [
                        {
                                label: 'ID',
                                name: '_id',
                                type: 'Number, min: 18, max:65',
                                transformation: 'function nome(val) {    \
                                        return val+" junior";
                                }'
                        },
                        {
                                label: 'Birthday',
                                name: 'birth_date',
                                type: 'Date, default: Date.now'
```

```
                },
                {
                        label: 'label number 3',
                        name: 'nome3_db',
                        type: 'String'
                },
        ],
        button : [
                {
                        label: 'Calculate',
                        name: 'button1',
                        function: 'function nome() {    \
                                return "hello";                 \
                        }'
                },
                {
                        label: 'Calculate2',
                        name: 'button2',
                        function: 'function nome() {    \
                                return "hi";                    \
                        }'
                }
        ],
        query: {
                age: {$lt: 20}
        }
}, //end index page

show : {
        row : [
                {
                        label: 'label row1',
                        name: 'nome1_db',
                        type: 'String',
                        transformation: 'function nome(val) {   \
                                return val+" junior";
                        }'
                },
                {
                        label: 'label row2',
                        name: 'nome2_db',
                        type: 'String'
                },
                {
                        label: 'label row3',
                        name: 'nome3_db',
                        type: 'String'
                },
                {
                        label: 'label row4',
```

```
                        name: 'nome3_db',
                        type: 'String'
                    },
                ],
                button : [
                    {
                        label: 'CalculateShow',
                        name: 'button1show',
                        function: 'function nome() {     \
                                console.log("Show");     \
                        }'
                    },
                    {
                        label: 'Calculate2Show',
                        name: 'button2Show',
                        function: 'function nome() {     \
                                return "helloShow";                    \
                        }'
                    }
                ]
        } //end show page

} //end collection

exports.collection = collection;
```

This is a complete DSL file that describes a collection, both its index view (where all the documents contained in the collection are displayed) and its document view (where you can see a single document).

So lets begin with the basics:
In the first line you declare you want to describe a collection and you want to set its name and where to get the data, therefore

```
collection = {
        label: 'Awesome things',
        name: 'notsoawesome',
        position: 3,
```

The label field is where you specify what name you want displayed on the web pages.
With MaaP you can rename almost everything you want without every worrying about mismatches, we'll handle all the dirty job. If you don't specify any label the original name will be used instead.
The name field specify what collection on your database you want to use as the source of your data, and lastly the position field specify the position you want this collection to be on the navigation bar menu of the browser.
If you don't specify any values MaaP will sort things by himself.

In the next piece we'll see how to set the index of the collection, also called collection view.

```
index : {
                populate: [{collection: 'nome_collection', key: 'chiave_document'}],
```

```
perpage: 20,
sortby: '_id',
order: 'asc',
column : [
        {
                label: 'Identification',
                name: '_id',
                type: 'Number, min: 18, max:65',
                transformation: 'function nome(val) {
                        return val+" junior";
                }'
        },
        {
                label: 'Birthday',
                name: 'birth_date',
                type: 'Date, default: Date.now'
        },
        {
                label: 'label coumn3',
                name: 'column3_db',
                type: 'String'
        },
],
```

"index:" tells MaaP that you're beginning to set the collection view.
The perpage field specify how many rows (hence how many documents) should be displayed on a single page. So if you have 120 documents and you set perpage to 50 you'll get 3 pages, two with 50 document each and one with 20.
The sortby field specify what is the column that gets sorted by default and, as you could probably imagine, order specify what order.

The column keyword marks the beginning of the columns definition. For each column you can specify a label, the source field from the database and its type. You can also specify some restrictions on the values, such as min and max.

The populate keyword allows you to define an array of collections from which you can get additional data.
For example, if you're creating a DSL for the Team collection and you want to also use data from the Coaches one, you simply need to use "populate: [collection: 'coaches', key: 'coach']".
You can them use coach.name and get the name of your football team coach next to the name of your team.
Another thing you can do is use a javascript function to transform data from your database before it get sent to the client. This is done using the transformation keyword. In this case we add " junior" to every name displayed.
You can write whatever you want with javascript, but bear in mind that the responsibility to keep the data coherent is on your shoulder and yours only.
Moving on we get to the queries

```
query: {
                        age: {$lt: 20}
```

```
                }
```

You can use queries to further restrict the data you display in the columns. In this case we select only the people younger that 20.
And at last we see how you can describe the view for your documents.

```
show : {
                row : [
                        {
                                label: 'label row1',
                                name: 'nome1_db',
                                type: 'String',
                                transformation: 'function nome(val) {    \
                                        return val+" junior";
                                }'
                        },
                        {
                                label: 'label row2',
                                name: 'nome2_db',
                                type: 'String'
                        },
                        {
                                label: 'label row3',
                                name: 'nome3_db',
                                type: 'String'
                        },
                        {
                                label: 'label row4',
                                name: 'nome3_db',
                                type: 'String'
                        },
                ],
```

The keyword show starts the document section of the DSL file.
The show sections offers the same functionality available on the index section, so there is nothing we haven't already seen.

7 Commands and options

7.1 Commands

The only command available on MaaP for now is create, which is the command used to create a new project.

7.2 Options

The following options are available:

- -h, –help : output usage information
- -V, –version : output the version number

- -N, –name [project_name] : specify the project's name
- -O, –output [output_path] : specify the output path (default is "./")

8 Issue and bug reporting

In the event that you have a problem with MaaP or find a bug, you are more than welcome to report it using the issue service of our repository at

https://github.com/ApertureSoftware/MaaP

We are also open to suggestion, so if you have an idea for a new features feel free to tell us.