MAAS

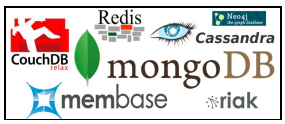
by Red Babel.

1 Introduction

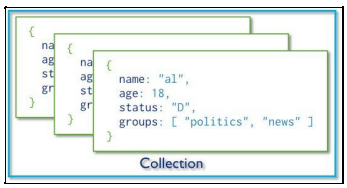
Nowadays there are many tools for making the job of the data administrators easier and more effective. Data are not only used by tech people, who make enquiries directly to the database, it needed to be accessible by business people. Their job is to use it in order to make business decisions. How do they access to it, if they cannot make queries directly to the sources where the data is stored? At the present day there are tools difficult to use by business people. This proposal describe our vision based on the experience gained. The aim is to produce a tool easily fruibile without programming skills.

During the last years the Information technology [IT] Industry started dealing with a huge quantity of data. This trend pushed computer scientists, engineers and system architects to change the existing data storage/retrieval and data visualization technologies. For 20 years the relational database model was the most important, and in most cases, the only one to be used in the industry. The main advantage of the relational model, and eventually its major flaw, was that it imposes an high level of organization of the data.

Along with the increasing of the quantity of data, the type of the data has become less structured. Relational databases and the traditional SQLs weren't enough to keep up with organizing this huge data variety. As often happens, as soon as the problem is clear, also the solution becomes clearer: new kind of databases were born: the "Not only SQL" [NoSQL] databases.



The NoSQL database systems give an efficient way to store and retrieve data that requires less restrictive constraints compared to the relational one. This approach simplifies the data modelling, facilitates horizontal scaling and allows a tighter control on the availability of the data. One of the most prominent examples of NoSQL databases is MongoDB[1], a document-oriented database. MongoDB uses JSON-like[2] documents with dynamic schemas, the format is called BSON[3]. It allows to make the integration of data easier and faster.



Below we describe a tool born following our vision that allows to easily build web pages based on MongoDB collections and documents.

[1] http://www.mongodb.org
[2] https://en.wikipedia.org/wiki/JSON
[3] https://en.wikipedia.org/wiki/BSON

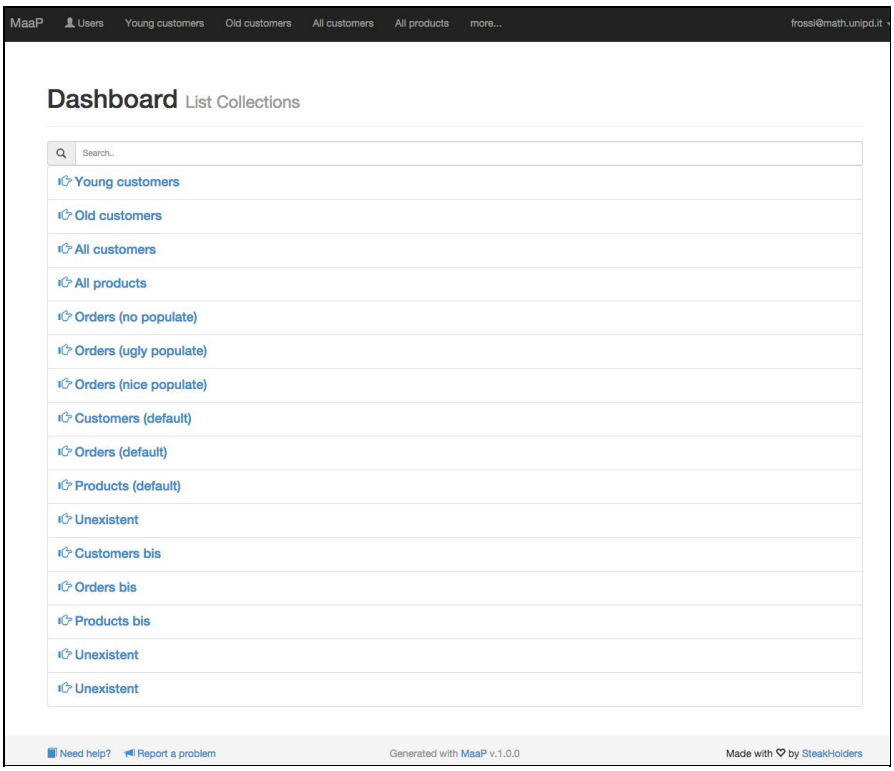
1.1 MongoDB as an admin Platform [MaaP]

Every application that uses a database to persist data requires: data administration and data visualization.

Data administration is a set of low level operations. They manipulate the objects that build the database, such as: tables, indexes and views. Normally these operations are performed by the system administrators or developers.

Data visualization, on the other hand, is an higher level of operation. Data visualization gives an interpretation of the business entities that reside in the data underneath. Usually, the final users who take advantage of this sorts of operations is everyone who needs to take business decisions based on the data of the system, the so called business people. As the name suggests, business people are expert of the business domain, and they usually don't hold any technical expertise.

MaaP elegantly resolves the data visualization problem. It is a flexible tool for the system administrators to extract and show graphical representations of the business entities from the data underneath. Its purpose is to simplify the implementation of UI for representing the business data by providing a domain specific language [DSL] that allows the user (the application admin) to design the contents of the web page with few clicks.



MaaP dashboard page with the list of all DSL definitions

MaaP was developed as a result of:

- http://www.math.unipd.it/~tullio/IS-1/2013/Progetto/C1.pdf.

Two MaaP implementations were developed in response to the cited tender:

- https://github.com/steakholders/maap;
- https://github.com/ApertureSoftware/MaaP.

It is possible to find a live demo of MaaP implemented by the SteakHolders at http://maap.herokuapp.com[4]. The source code for this demo could be found here: https://github.com/redbabel/maap.demo. Below you can find an example of DSL that has been used in the live demo.

```
collection(
        name: "customers",
        label: "Young customers",
        id: "youngcustomers",
        weight: 1
) {
        index (
                perpage: 20,
                sortby: "age",
                order: "asc",
                query: {
                        age: { $lt: 20 }
                }
        ) {
                column(label:"Id", name:"_id")
                column(
                        label: "Nome",
                        name: "fullname",
                        selectable: true,
                        sortable: true,
                        transformation: function(val) {
                                return val+" junior";
                        }
                )
                column(name:"email", sortable: true)
                column(name:"city")
                column(name:"age", sortable:true)
                column(label:"N. of orders", name:"orders", sortable:true)
        }
        show {
                // Default
        }
}
```

DSL for the *youngcustomers* page.

This code, once interpreted by MaaP, will produce two kind of pages: 'collection-index' and 'collection-show'. Collection-index specifies how to visualize the list of all the documents in a table format. 'Collection-show' specifies how to visualize a single document for that collection. A detailed explanation of the examples could be found here: https://github.com/steakholders/maap/wiki/DSL-Configuration-File-Example.

[4] username: 'frossi@math.unipd.it' - password: 'automi'.

Young customers Index Page

Id	Nome	email	city
533406891734c2fe5104024b	Vaibhavi Anglin junior	LRote@vestibulum.net	Laredo
533406891734c2fe5104024c	Gordon Nimon junior	JHughesdkaiya@pretium.com	Saugus

Graphical output for the 'index' section of the '*youngcustomers'* DSL,
page collection-index. Link to the demo:
http://maap.herokuapp.com/#/collections/youngcustomers

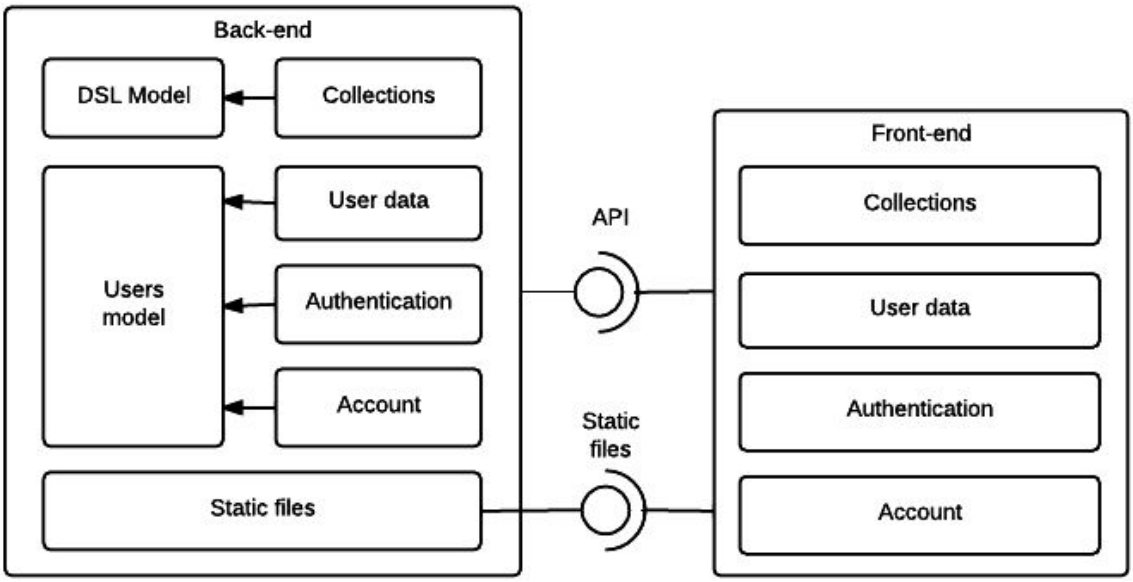
Document 533406891734c2fe5104024b

_id	533406891734c2fe5104024b
avatar	http://www.filltext.com/image/50/50
city	Laredo
email	LRote@vestibulum.net
orders	0
age	0
fullname	Vaibhavi Anglin

Graphical output for the 'show' section of the '*youngcustomers'* DSL,
page collection-show. Link to the demo:
http://maap.herokuapp.com/#/collections/youngcustomers/533406891734c2fe5104024b

Architecture

The two original implementations (StakeHolder and Aperture Software) share the same spirit and almost the same DSL definition. Both versions use a common server/client pattern. In both implementations the server is built in Node.js[5] and provides a set of REST[6]-like application programming interface [API]. Both clients are web clients built in AngularJS[7].



Main architectural components

The main difference between the two implementations is the underlying technology for the DSL interpretation.

Aperture Software implemented a parser for the DSL using PEGjs (http://pegjs.org/). The DSL definition could be find here: https://github.com/ApertureSoftware/MaaP/blob/master/maap_server/modelServer/DSL/DSLParser.js.

StakeHolder DSL uses macro to extend Javascript via Sweetjs (http://sweetjs.org). The DSL definition could be find here: https://github.com/steakholders/maap/wiki/DSL-File-Configuration.

Both approaches have pros and cons that will be not discussed in this document.
MaaS will be required to use the SteakHolders implementation as reference. The main architectural choices remain the same (e.g. in Node.js for the server and Sweetjs for the DSL definition).

[5] https://nodejs.org
[6] https://en.wikipedia.org/wiki/Representational_state_transfer
[7] https://angularjs.org

1.2 MongoDB as an admin Service [MaaS]

This project aims to build a web service that embeds MaaP and makes it available directly via the Web for multiple companies. Hints as to the intent of this project can seen in section '*Ramo servizio: MongoDB as an admin Service (MaaS)*' of chapter '*Requisiti opzionali*' of the original proposal.

MaaS extends MaaP in two meaningful dimensions: Software as a Service [Saas] and [DSL].

Dimension SaaS

MaaS should be built as a multi-tenant application architecture (i.e. a single instance that serves multiple tenants). A tenant is a group of users who share a common access with specific privileges for each MaaS instance. A multi-tenant architecture MaaS provides, to every tenant, a dedicated share of the instance including its: data, configuration, user management and tenant individual functionality.

Dimension DSL

MaaS should allow online editing of the DSL definitions. This feature is the natural extension of MaaP to "as a Service" model. It should also add new concepts on the DSL itself like the concept of Dashboard and Action.

The following two sections describe the SaaS model and the extensions of the original DSL.

2 The SaaS Model

The market offers many tools for database administration, either for relational DBs (phpMyAdmin for MySQL http://www.phpmyadmin.net, pgAdmin for PostgreSQL http://www.pgadmin.org/) and NoSQL DBs (http://docs.mongodb.org/ecosystem/tools/administration-interfaces/). For all of those tools, the final user is the Database Administrator (DBA) or a software developer. Generally speaking these tools are developed for tech people and not for the business ones.

In order to resolve this problem, there are some technology stacks that use high level programming languages. These languages allow the developer to rapidly create tables and effective data visualization for business purposes. On the other hand, the installation and the maintenance of these tools represent a great burden for the DBA. In many cases it is so cumbersome that it requires too much knowledge in different areas, that it becomes impossible to maintain everything.
The SaaS model tries to overcome this problem. One of the biggest selling points for SaaS services is the potential to reduce IT support costs by outsourcing hardware, software maintenance and support.

2.1 Real world use case

Let's say that a company called Pied Piper wants to use http://maap.herokuapp.com/ for visualizing the data of its own database. Richard, the CEO of the company subscribes to MaaS and, among all the other personal data, he inserts the name of the company. Richard is now the owner of the Pied Piper's account. Once his email is verified and he logins into the system he can invite other users. He invites his CTO, Dinesh, as admin of the account. Soon after he invites Jared, Pied Piper's business developer with the role of member. Finally he invites Monica, as guest user, who represents the investor of Pied Piper.

One day, Richards cannot login into the system anymore. He contacts Bighetti, from the customer support at http://maap.herokuapp.com/, to ask for support, hoping that Bighetti can resolve his problem. Bighetti logins with his Super admin user capabilities and after a brief inspection he manages to resolve Richard's problem.

The use case above describes a typical usage of the SaaS model in MaaS. We now dig deeper into the details that make everything work.

2.2 Entities

All the domain entities presented in the example above are described in the SaaS model of MaaS. The SaaS model defines each entity and the relationships between them.
The entities that form the SaaS model are:

1. Companies;
2. Users;
3. Super admin;
4. Roles and capabilities
5. DSL definitions;

Companies

A company is an independent entity from other companies. Many users belongs to a company. Every user must belong only to a single company.
For example, if Richard wants to use MaaS for his personal project (ex: Hooli), he has to subscribe to http://maap.herokuapp.com/ with a different email and create a different user. Each company can visualize data from multiple databases.

Users

Every user who belongs to a company has one role and each role has a set of capabilities. The roles are the following: Owner, Admin, Member, Guest.
Every capability is allowed only inside the company where the user belongs to (e.g. the owner of Pied Piper cannot disable the user of Hooli).

Super admin

The main difference between a Super admin and a user is that the first ones don't belong to any company. This kind of users is thought for the system administrators who have to manage the whole system and provide support to the other users.

Roles and capabilities

Below are specified the minimum capabilities for each role. The guest can only execute the DSL he can access to. The member can create DSL specifications, execute/read and write DSL created by himself and the DSL the administrator gives him the right to.

The admin can do everything with all the DSL owned by the same company. In addition he can also invite/disable/change roles of/set the access to DSL specs to the other users. He cannot disable the owner of the company.

The owner of the company can do what the admin can do. He cannot be deleted. The Super admin is the administrator of the entire MaaS. He can perform all the operations to manage the companies, he cannot be deleted and he can impersonate other users inside the platform.

DSL Definitions

It is possible to perform 3 operations for each DSL definition:

- Read, the user can read the DSL definition;
- Write, the user can create a new DSL or update an already existing one. The user can edit the DSL specifications using the browser.
- Execute, the user can visualize the data as result of the corresponding DSL definition.

3 Domain Specific Language

MaaP vs MaaS

In MaaP the two main pages were defined as Collection-index and Collection-show. In MaaS the original main elements shall be present, although with different names and a slightly different logic. Two additional kind of pages must also be present: Cell and Dashboard. Each DSL definition has one and only one of the following main elements as root. Every main element defined in a DSL definition must have an identification name. DSL definitions cannot be shared among different companies.

The main elements are: Cell, Document, Collection, Dashboard.

Cell

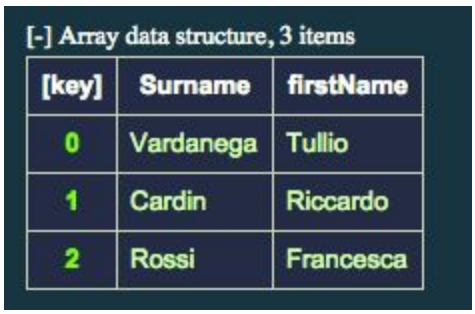
Cell is the main element that visualizes a single value. The value visualized can be: String, Number, Link, Image, Date.
A cell can take as input the result of a query in the specified mongo collection. The results can be ordered by a specific attribute field. The value visualized is the the first value of the results set.
A cell can take as input also an arbitrary value not linked with a query.

Document

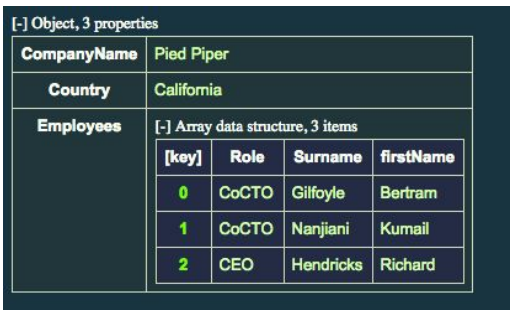
Document corresponds to the Collection-show entity in MaaP. In MaaP the DSL description of a document was always included inside a Collection-index. In MaaS a Document could be also declared independently by the other entities. The same requirements applied in MaaP for the concept Collection-show shall be applied to *Document*.

In addition to the MaaP's attributes, MaaS should take also care of four types of data: Array, Object, Link, Image. The concept of Action should also be introduced.

- Array/Object. A proper graphical visualization should be rendered. Here there are two examples: http://jsoneditoronline.org and http://chris.photobooks.com/json;


[-] Array data structure, 3 items

[key]	Surname	firstName
0	Vardanega	Tullio
1	Cardin	Riccardo
2	Rossi	Francesca


[-] Object, 3 properties

CompanyName	Pied Piper
Country	California
Employees	[-] Array data structure, 3 items

[key]	Role	Surname	firstName
0	CoCTO	Gilfoyle	Bertram
1	CoCTO	Nanjiani	Kumail
2	CEO	Hendricks	Richard

Graphical example for an Array Graphical example for an Object

- Link. A web link with a classic behaviour (e.g. open a new window/tab);
- Image. An image should be rendered. It should be possible to specify appropriate attributes such as width and height.

Action

An action specified in a Document provides as implicit input, the document itself. The actions are run server side and only a configuration is given. Only a specified set of built-in actions could be chosen. The addition of a new action is not meant to be done by the end user. A new action can be added only by the administrator of the platform.
Two built-in actions must be provided: 'export' and 'sendEmail'.

1. Export. The system exports the current Document 'csv'[8] and 'json'[9].
2. Send Email. The system sends an email to a specified address. The smtp config values are global and provided by MaaS, they are not company specific.

Collection

The same requirements given in the MaaP's Collection-index are applied to *Collection*. Collection is specified by five core concepts:

1. Identity of the specific collection (e.g. name and label)
2. What subset of documents we want to see (e.g. query);
3. What subset of attributes we want to see when we show the list of documents (e.g. index and row);
4. What subset of attributes we want to see when we show a specific document (e.g. show and column);
5. Action, applied to the whole collection.

Row and Column are very similar in nature and share most of the properties. Among the original properties the concepts of Array, Object, Link and Image should be introduced as seen in Document.

Action

An action in a collection works in a similar way as in a Document. The main difference is the implicit input. It should be possible to define if the actions will have the whole collection as input or only the visualized subset. The same requirement as Action under Document should be applied. The same two built-in actions should be provided ('Export' and 'SendEmail').

[8] https://en.wikipedia.org/wiki/Comma-separated_values
[9] https://en.wikipedia.org/wiki/JSON

Dashboard

A dashboard specifies a grid layout. The layout is defined by subelements called ‘row’. ‘row’ describes a new row in the grid layout. The columns inside a row are implicitly defined by the number of main elements contained in ‘row’ itself. Dashboard can be declared only as root element of the DSL.

Here is an example:

```
dashboard( // 2 rows
        row( // first row with 1 columns
          cell()
        )
        row( // second row with 3 columns
          collection()
          document()
          cell()
        )
){
  name: ‘pippo’
}
```

5 Requirements

5.1 Minimum requirements

Below are specified the minimum requirements to satisfy:

1. Realization of MaaS. The SaaS and the DSL dimensions as described in this document are required.
2. The technological stack includes:
    a. Node.js for the backend. MaaS should target the Long Term Support [LTS] version Argon[10];
    b. MongoDB, version >=3.x as the application database;
3. The system should be built using a high-level Node.js framework. We strongly advise to use IBM Loopback (http://loopback.io);
4. The system must be deployable to Heroku (http://www.heroku.com). Heroku is a cloud platform as a service [PaaS] supporting several programming languages. Deploying applications to Heroku generally means pushing an app to Heroku using Git (http://git-scm.com). Git is a distributed version control system for software development.
5. The source code must be published and versioned using either: github (www.github.com) or bitbucket (www.bitbucket.com)

5.2 Optional Requirements

React branch

The original MaaP uses Angular.js as front-end framework of choice. Angularjs implements the well known design pattern called MVC and is created and maintained by Google.
React.js (https://facebook.github.io/react) is a framework developed by Facebook that, together with Flux (http://facebook.github.io/flux), tackle the same problem with a different perspective.
React-native a framework for building native apps (e.g. iOS or Android) using React. With React-native is possible to reuse React components to build mobile applications.

Requirements for this branch are divided in two main topics:

1. Use React/Flux to implement a component based front-end for MaaS instead of Angular.
2. Develop a mobile app for MaaS using React-native.

[10] https://nodejs.org/en/blog/release/v4.2.0

Language branch

MaaP does not provide any help for creating the DSL specifications. In MaaS, the editing of the specifications will be performed by the user on the web browser. In order to simplify the creation of the DSL specification, the editor in MaaS should provide some common features already present in many editors such as: syntax highlighting, autocompletion, indentation, snippets, etc etc.

5.3 Change requests

Requirements changes are not allowed, unless they represent a clear improvement of which is specified in this document. The customer may make some requirement changes, either before lodging or during the implementation of the product.

5.4 Documentation

Along with the submission of the implementation of the system, it has to be submitted: the user manual and every necessary documentation that allows the customer to use the system. The documentation must be written in English.

5.5 Warranty and maintenance

The supplier has to demonstrate during the R.A. (*Revisione di Accettazione*) that the product works correctly and accordingly to the specified requirements.
Fixing bugs and flaws not compliant to the requirements are entirely at the expense of the supplier.

5.6 Credits and Licence

RedBabel has interests into this project as proof of concept using the technologies specified above. The system will be distributed under the MIT licence, the supplier will be mentioned in the copyright credits. Also RedBabel will be credited under the section *credits* in the README file. Please refer to the Italian Law about the management of public bids for what is not specified in this technical specifications document,

6 The proponent

RedBabel (http://redbabel.com/) is an IT consulting firm based in Amsterdam formed by Alessandro Maccagnan and Milo Ertola. The firm operates in Amsterdam where it holds close relationships with the startup ecosystem of the city. Milo and Alessandro are Rockstart Alumni and advisors at Coffeestrap BV (http://coffeestrap.com).
RedBabel contacts:
Alessandro Maccagnan (reference contact for RedBabel): alessandro@redbabel.com
Milo Ertola: milo@redbabel.com

7. Appendix

7.1 Rest Architecture Style

The designing of the MaaS platform must comply to a set of architectural principles that goes under the name of REST. REST is a convenient way to conceive, design and implement modern web apps.

We suggest to adhere as close as possible to those principle. Since REST is a style there are many different implementations and flavours that implement it. The supplier can take advantage from the huge offer he can find on internet. Wikipedia and Stackoverflow are also good sources for better understanding the problem.

We leave a very brief list with the main products that implement the REST style and they are widely used in the industry:

- IBM: http://loopback.io/
- WalMart: http://hapijs.com/
- PayPal: http://krakenjs.com/

7.2 Security

Security is a central issue for the development of SaaS product. Below are listed some key points in order to implement the security of the service.

Invitation flow

As anticipated above, except for the company's owner, the other users can create an account only when they are invited.
Let's see the invitation flow more in deep:

- Richard, the CEO of Pied Piper, creates the company's account. In the signup form, he puts his company's data and his email address;
- At this stage of the signup process he invites the other components of his team, the first one is Dinesh, his CTO;
- The system sends an email to Dinesh containing a link that let Dinesh to access to the signup page. The link contains a JWT[11] token and it is associated to the invited user. If Dinesh doesn't have time to signup to http://maap.herokuapp.com/ the link is valid up to a configurable amount of time;
- Once Dinesh clicks the link, he will be redirected to the signup page where he will put infos, password etc etc.

[11] http://jwt.io/

Users and Super admin credentials

A user (and a super admin) can change his own password with the usual procedure of password reset. The user receives an email with the reset password link. Similarly to the invitation flow, also the reset password link contains a JWT token.

All the passwords must be persisted encrypted at any time . The encryption library we suggest to use is scrypt (https://www.npmjs.com/package/scrypt).

Login

The user can access to the system using the standard email/password combination.

7.3 Performance

Dealing with data visualization can lead to performance issues, especially when there is a huge amount of data to transfer and/or to render in the web page. In order to handle this issue, we suggest to consider some principles before designing the system.

Only Requested Resource Principle (O.R2.P.)

Before implementing the REST API, please consider the possibility that a huge amount of data could be requested by the client. The current implementation of MaaP receives all the data from the database and then it does the rendering. This approach can crash the client if the data involved is too big. Don’t send more data than what is strictly requested from the client (ex: use list pagination).