Functional Requirements Document

### Introduction

To allow the municipal workers of the city of Gotham to have access to some days off to recuperate, a time management application need to be created.

#### 1.1 Purpose

The purpose of this functional and technical requirements document is to provide documentation to a “Time Management solution” required by the town hall of the city of Gotham to set up and operate a time management application for the municipal workers.

#### 1.2 Scope

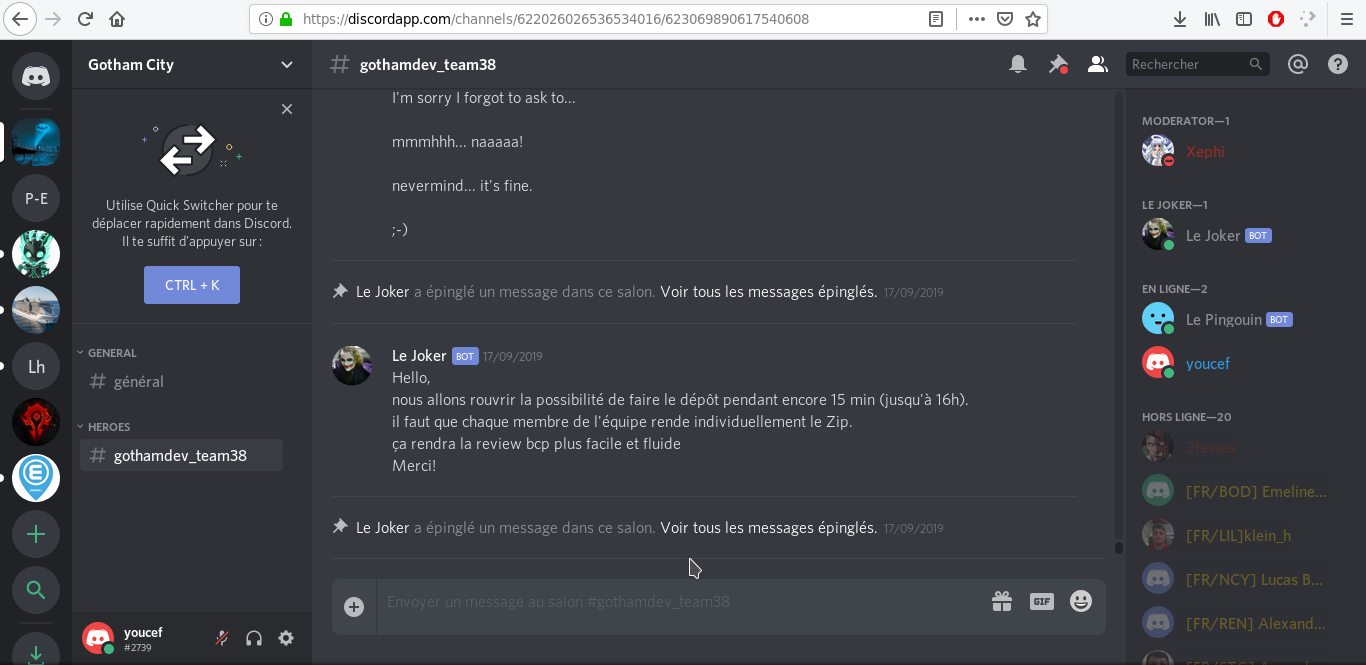
This functional and technical and technical requirements document will outline the functional, performance, security, UX and other system requirements identified by the town hall of the city of Gotham to implement and operate a time management application.

#### 1.3 Background

The town hall of the city of Gotham’s team, being themselves municipal workers, acknowledge the impact that days off have on the performance of the workers. Consequently, the team has set the following strategic and tactical objectives. Our goal is to build a time manager application allowing workers to report their departure and arrival times, and managers to view averages of the daily and weekly hours of the team over a given period.

#### 1.4 References

Other files will be attached to this document where you can find further description of the functioning of the application:

* **Wireframe-time-manager***.pdf* : A document that defines the organization of elements and forms.
* [**T-POO-700\_project**.pdf](https://gandalf.epitech.eu/pluginfile.php/312/mod_assign/introattachment/0/T-POO-700_project.pdf?forcedownload=1) : Original specifications.
* [**https://discordapp.com/channels/622026026536534016/623069890617540608**](https://discordapp.com/channels/622026026536534016/623069890617540608) : Discussion with the CEO

#### 1.5 Document Overview

This document will firstly list the functional requirements, then the other types of

requirements that are mainly linked to the development process. All together, they describe

functionalities that the development must answer with a technical solution.

### Functional Requirements

#### 2.1 Context

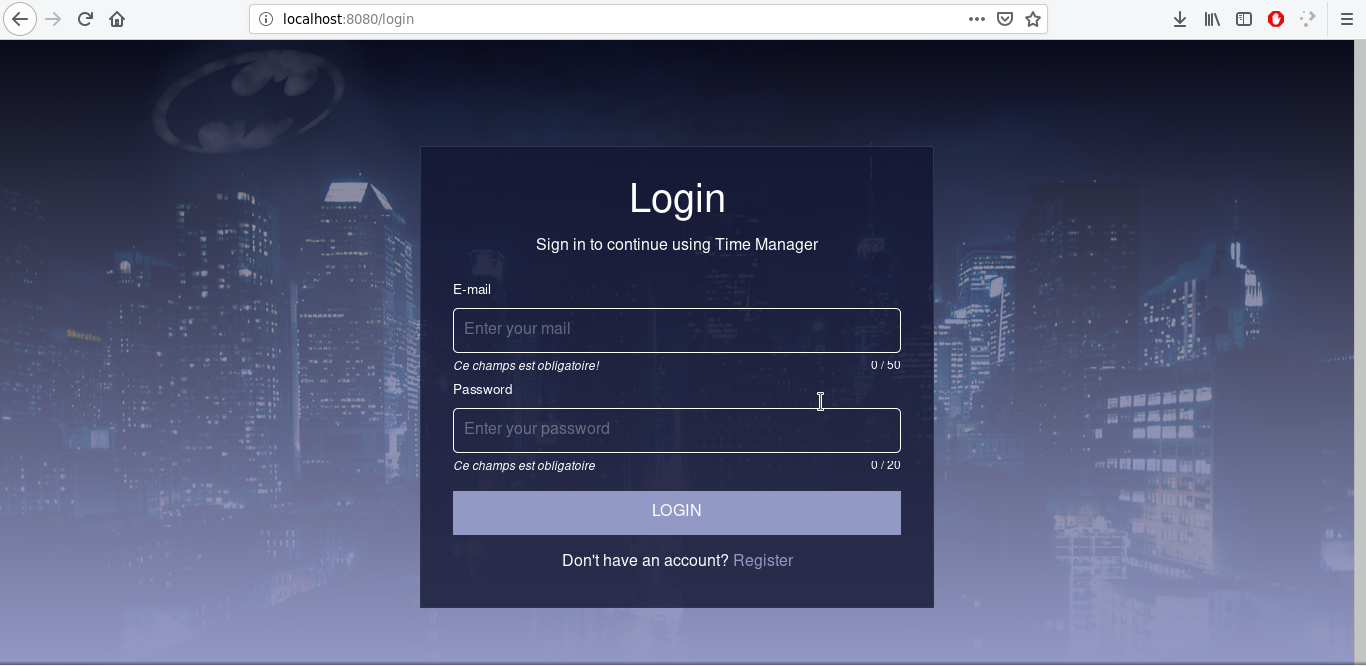
To develop an application that answer the needs correctly, we discussed with the clients and the users to define more detailed requirements.

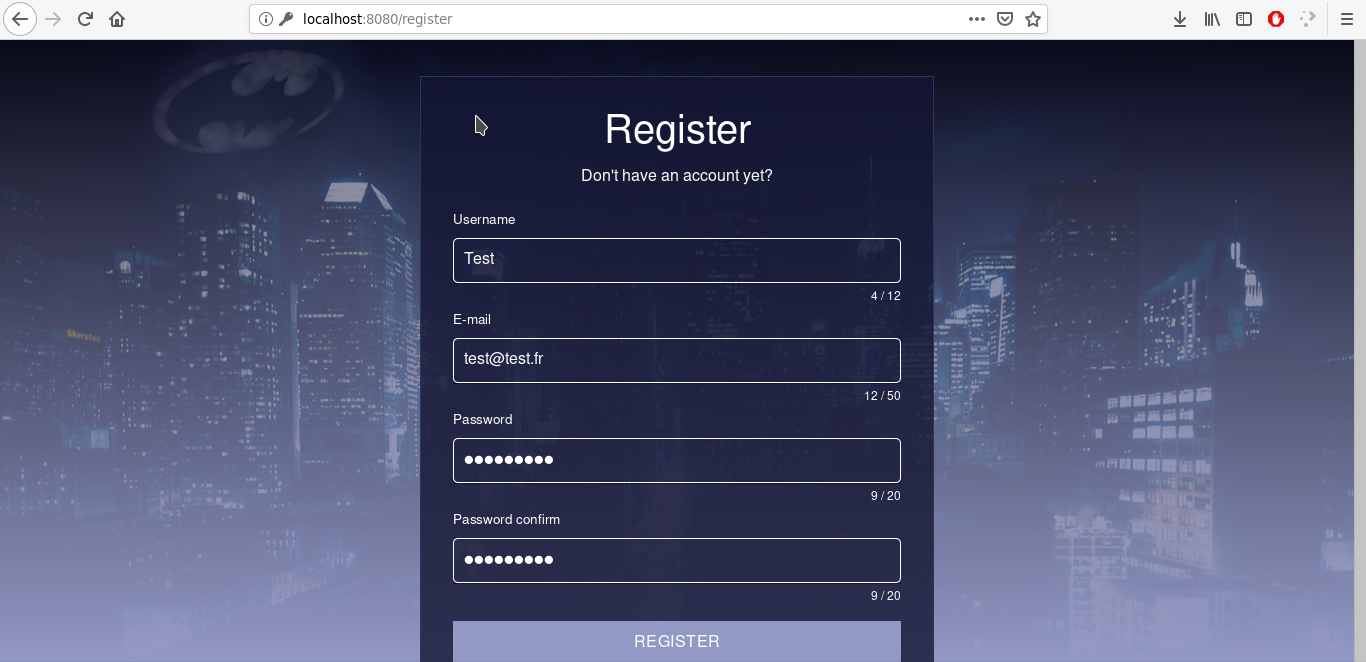
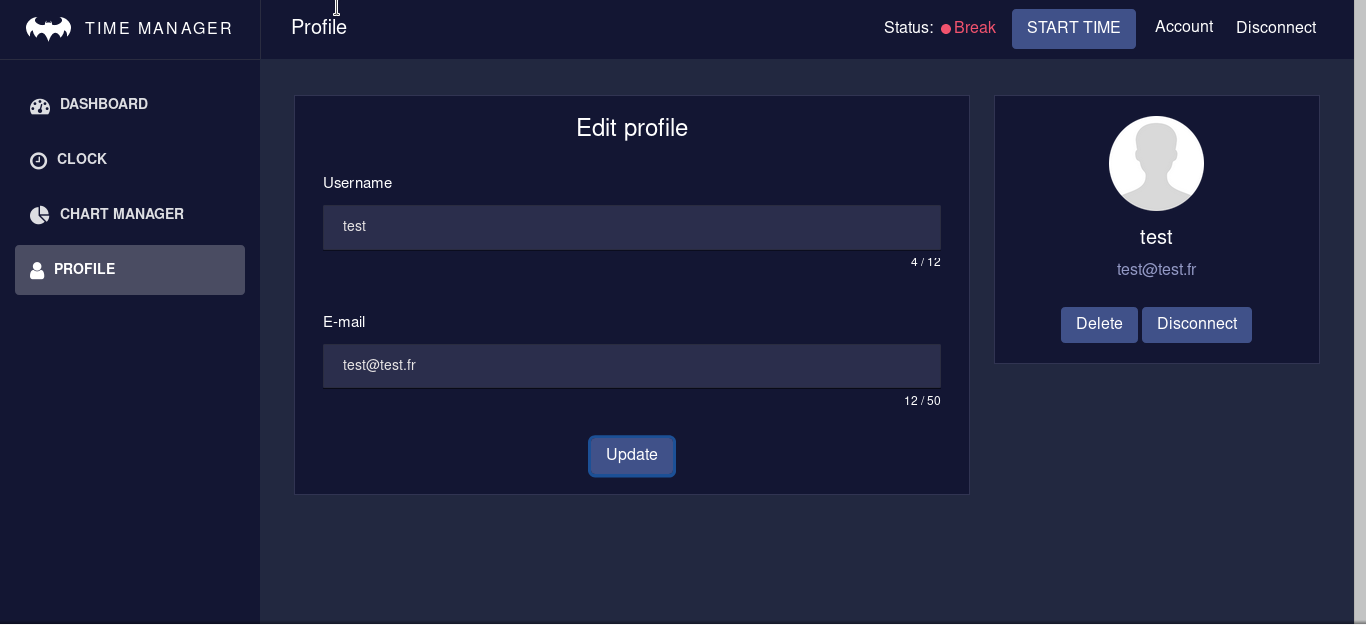
#### 2.2 User Requirements

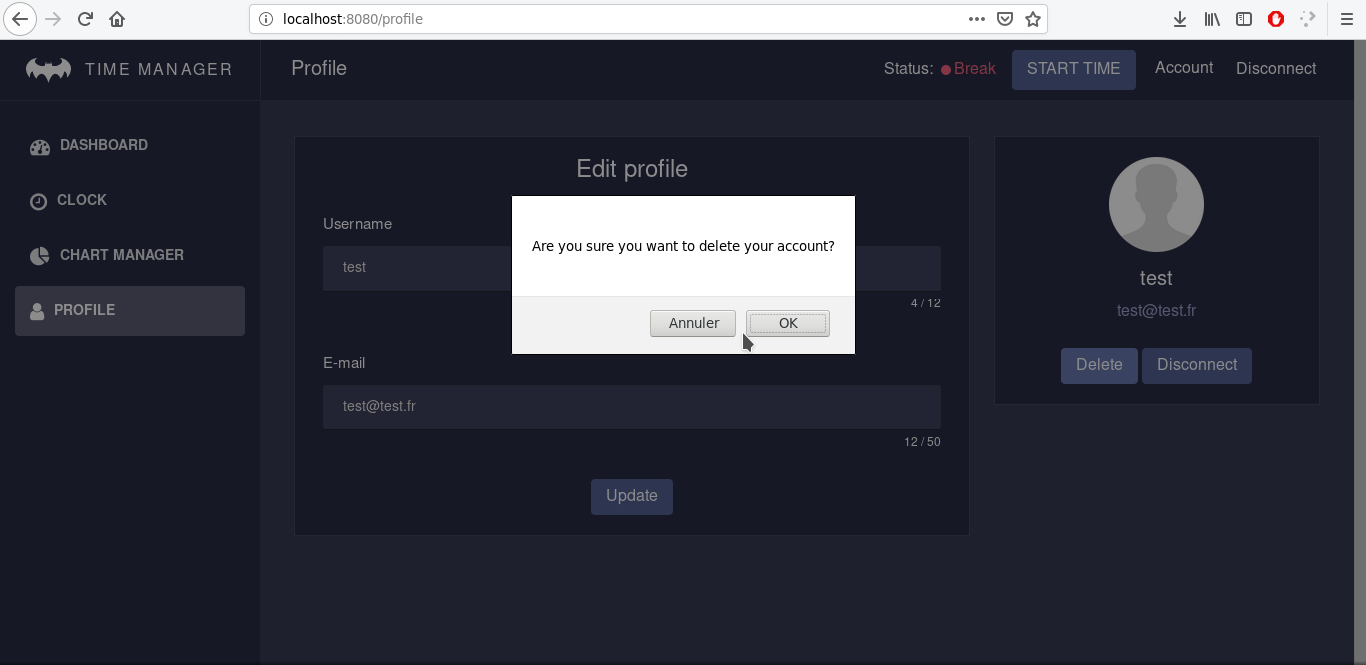
We were asked to group the users in 3 types with different rights and actions:

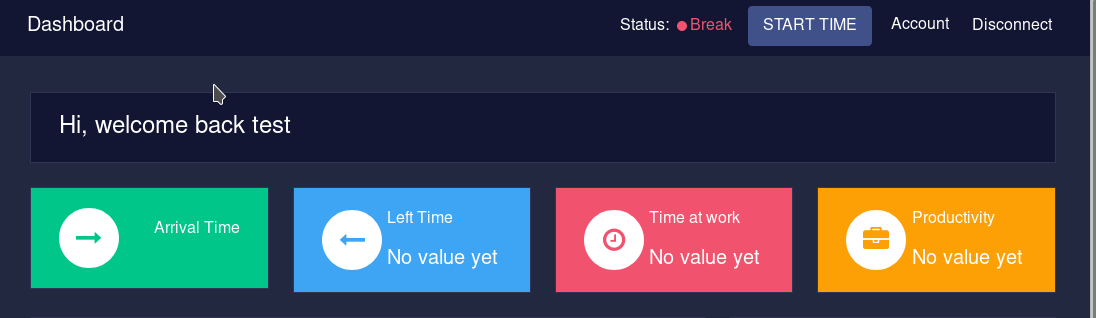
**Employees:**

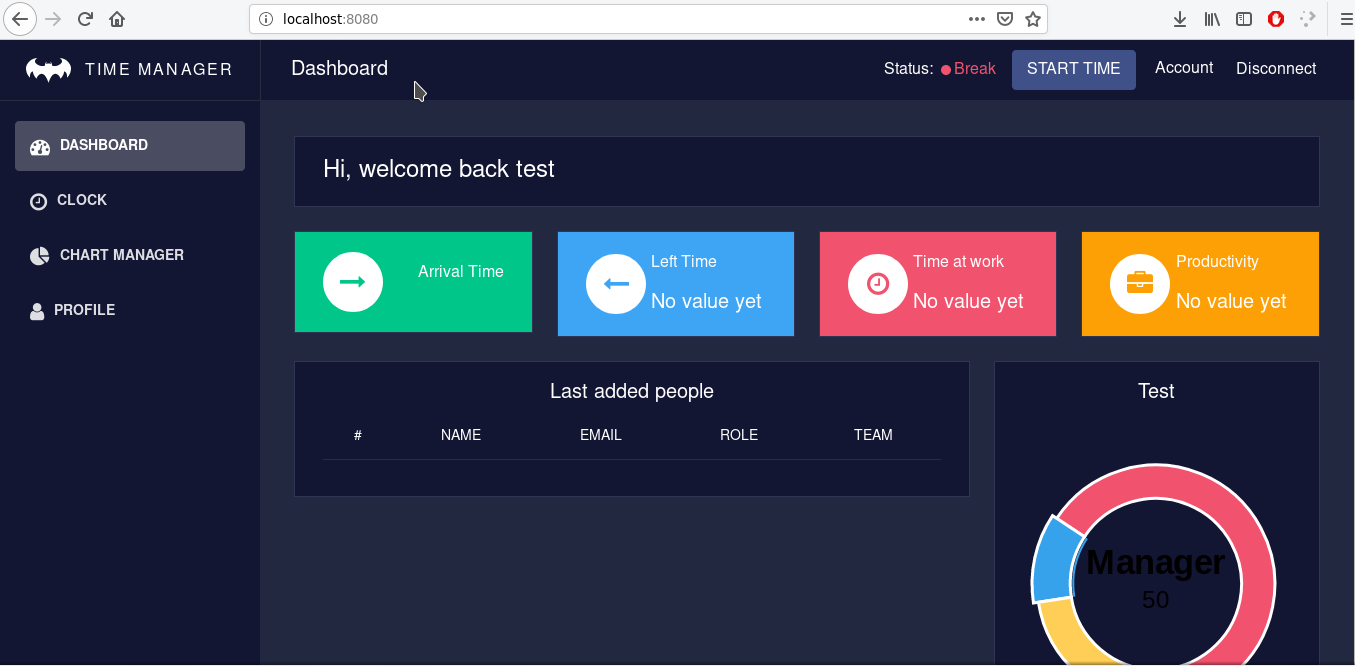
* Log in / out: Login system with email and password



* Registration: Account creation form
* Edit account information: Form to edit
* Delete account: Button, confirmation box.



* Report working times: Create, edit, delete working times
* See a personal dashboard (Overview of personal data)



**Managers:**

* Everything that an employee can do
* Manage teams of employees

1. Create and delete teams, assign users to it

* View the averages and total of the daily and weekly hours of the team over a given period

1. Displaying graphs

* View the daily and weekly working hours of an employee over a period of time

1. Displaying graphs

* View their employees’ dashboards
* Assign ‘should’ work times to employees

**Admins:**

* Everything that a manager can do
* Promote a user from the rank of employee to manager
* View the dashboards of all users
* Delete accounts of all users

#### 2.4 Functional Requirements

The functional requirements can be grouped in following themes:

* Manage working time billing
* Help users with the basic functionalities
* Inform the user

##### 2.4.1 Functional Requirements - Working time and billing:

The main reason a company is able to survive is because their customer pays their service

or product. To maintain a good relation with the customer, it is important to make him pay

only for what he gets. Therefore, it is needed to follow the working time on each project

and to separate it from the working time like administration.

##### 2.4.2 Functional Requirements -Helping users:

The user may need some help at some point. If none is provided or not easy to access the

user will not try further and leave the application. To avoid this, we will add an help section that will be easy to find. The user will be able to find informations on the basic usages of

the application.

For more complex operations (like those for managers and admins) it will be possible to

make some meetings to explain the concerned users how to make those operations.

##### 2.3.3 Functional Requirements – Inform the user

The user must not be lost in the application, not knowing what is happening. For this

reason, the design must not be too complex, and things must be easy to find. Once the user

can orient himself in the application, he must be aware of the consequences and effects

of his action. Therefore, the user must have feedback when his actions have been

processed, erroneous or not allowed.

### Other Requirements

#### 3.1 Interface Requirements

The accessibility and the interface of the application is very important. It strongly impacts

the way a user sees the application and his feeling about it. Therefore, the application must

be responsive, so it can also be used on mobile and on small computer screens

No other devices than a computer of mobile phone must be needed.

The backend must be reachable through internet, on every public network.

##### 3.1.1 Hardware Interfaces

The bare-metal server has one IPv4 address and one IPv6 address correctly routed on

Internet.

The physical ip addresses are:

* IPv4:
* IPv6:

The specifications of the server is:

* CPU:
* RAM:
* DISK:
* OS:

3.1.2 Software Interfaces

The application is embedded in multiple docker images, all those images communicate

together to fulfil all required functionalities of our app.

Those are our docker images used:

* API

1. Based on: Elixir Phoenix framework
2. What to do: Copy our backend project. Get and install dependencies
3. Once started: Launches the backend server

* Database postgresql

1. Image: postgres:10
2. Used to: Setup and start the database to store our application data

* Web interface

1. Based on: VueJs
2. What to do: Install all necessary packages, copy our frontend project in it
3. Once started: Launches the server hosting our interface

* Mobile interface

1. Based on: Cordova & VueJs
2. What to do: Install all necessary packages (Android, IOS platform ….)
3. Once started: Build and run on your device or project

##### 3.1.3 Communications Interfaces

All services of the system communicate via local network of docker network.

Exposed ports on internet are:

* For the API: 4000/TCP
* For the Web Interface: 8080/TCP
* For the DB interface: /TCP The database will not be exposed to the internet network for security reasons.

A database web interface is exposed to the internet network, for managing database correctly for system administrators, this interface will be completely secured by a login/password, and is not be accessible for anonymous user.

#### 3.2 Hardware/Software Requirements

The system will be hosted in a bare-metal server, with multiples services in containers

with Docker:

* 1 service for expose the api system
* 1 service for the expose the web interface
* 1 service for run the postgres database for host the all datas

All the system will be set up and linked each other in the same server. If the server is

down, all the system is down (api, web interface and database).

#### 3.3 Operational Requirements

##### 3.3.1 Security and Privacy

In case of loss or corruption of data: there must be a database backup in order to recovery

a working state. A backup must be made frequently enough to limit loss.

In case of disclosure of secrets or sensitive information, of Disclosure of privileged/privacy

information about individuals or Corruption of software or introduction of malware, such

as viruses: stop the application and remove it from the network. Develop a fix and apply

the changes to the production environment. Once everything is up again, identify the source and consequences and prepare measures to avoid recidivism.

##### 3.3.2 System Availability

The application must be available to users of the enterprise in office hours, without downtime.

The maintenance can be performed in the night of each day in the week, out of the office hours specified above.

##### 3.3.3 Data Retention

The system will retain application logs and access records information for 2 years, for legal reasons. Database records is fully able to be deleted at any time.

##### 3.3.4 Error Handling

Error handling is managed by the API service. If an error occurred, the service

continues to run normally. Error will not stop the service, it’s mandatory for avoid

downtime of the system.

An error system tracking will be in place in the future for alert administrators if an

error occurred.

##### 3.3.5 Validation Rules

There are only validation rules for user's creation and update.

Theses validations are:

* **Email:** Required, and a valid email with format : test@test.test.
* **Username:** 4 characters minimum and 12 maximum. Accepts all characters.
* **Password:** 8 characters minimum and 20 maximum. It must contain at least a lowercase character, an uppercase character and a digit.