

Software Design Description

BoatRegatta

5ISS

ABSTRACT: THIS DOCUMENT EXPANDS THE FUNCTIONALITY DESCRIBED BY THE FEATURES IN THE **SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS) v0.2**. IT DESCRIBES A PART OF OUR **GPS** TRACKING SYSTEM FOR BOAT REGATTAS. THE PURPOSE OF THIS DOCUMENT IS TO GIVE A DETAILED DESCRIPTION OF THE REQUIREMENTS OF THE **IoT** TRACKING PROJECT. THIS PROJECT AIM IS TO DEPLOY AN AD-HOC **LoRa** NETWORK FROM SCRATCH AND DEVELOP A USER-FRIENDLY WEB APPLICATION TO ALLOW PEOPLE TO FOLLOW THE EVOLUTION OF BOAT REGATTAS IN REAL TIME.

KEYWORDS: **LoRa**, **GPS**, BOAT REGATTAS, DEPLOYMENT OF AN AD-HOC NETWORK, HARDWARE, WEB APPLICATION.

AUTHOR	PROJECT MANAGER	PRODUCT OWNER
ALVAREZ Josué	ANAK STELL Cyril	INSA Toulouse
ANAK STELL Cyril		
DIOP Mame Aminata		
DUTHOIT Cécile		CVRL association
MJELSTAD Linn		
OUEDRAOGO Clovis		



Revision History

Version	Date	Author	Change Description
1.0	18/12/2016	Josué ALVAREZ	Structure
1.1	21/01/2017	Clovis Ouedraogo	Gateway to LoRaWAN
1.2	21/01/2017	Josué ALVAREZ	Deployment
1.3	23/01/2017	Cécile DUTHOIT	Layout
1.4	23/01/2017	Linn MJELSTAD	Rereading

TABLE OF CONTENTS

System design	6
Architecture components overview	6
Architecture constraints	7
General constraints	7
Network constraints	7
Web Server constraints	7
Web Application constraints	7
Android Application constraints	7
Device constraints	8
Architecture interfaces	9
Device to Gateway	9
Gateway to LoRaWAN	10
System schematic and definitions	10
Upstream protocol	11
LoRaWAN to Data Subscriber Service	14
Services to MongoDB	15
Data Provider Services to Android and Web Clients	16
Device Monitoring Service to LoRaWAN server.	17
User interfaces	18
Web Application	18
Android Application	18
Detailed component design	19
Web server	19
Node.js server	19
Database schemas	19
Auto-generated raw data services	23
Other services	23
Composite services	24
Buoy service	24
Data Subscriber service (MQTT client)	25
Glassfish server	26
ImportCSV	27
ParseCSV	27
ImportRacers	27
ExportRacers	27
	4

Web client	29
Architecture overview	29
Microservices	30
Main components and routes	31
Staff member and administration views	31
Anonymous user views	31
Android Client	32
Use case	32
Network	36
Device	37
System deployment	38
Software configuration and deployment	38
Deployment constraints	38
Deployment overview	38
Software configuration	39
Apache - Nginx Server	39
IoTracking VM	39
Components	39
Network interfaces	40
Configuration	40
LoRa Server VM	41
Network interfaces	41
Annexes	41
Traceability Matrix	41

I. System design

A. Architecture components overview

The main goal of the system described here is to track devices on boats during regattas, and to offer a way of viewing the races in both live and recorded mode. To achieve this, we provide a system that includes GPS tracking devices, a local network deployment, a web server and two clients.

The devices are responsible for retrieving GPS data and sending them through the network to the Web Server.

The network is composed of gateways which constantly scan the network to catch packets coming from the devices. Once the gateways get a packet, they send it to the LoRaWAN server which publishes the data to the web server via a Data Subscriber Service. These gateways communicate to the devices using a LoRa network, and to the web server through a specific long-range WiFi network.

The web server holds a database and a set of services. The Data Subscriber Service is responsible for gathering the published data from the LoRaWAN server, and inserting this data in the database. The Data Provider Services are a collection of web services (REST / SOAP) that expose the read-write database data to the clients.

The web client has two main goals: First, it is used to perform database administration, registration of the devices, races, and competitors. Second, it provides live or recorded viewing of the races.

The android client is used to locate, using GPS, the buoys that delimit the race map. It will send their location to the database after having acquired the position of all the buoys.

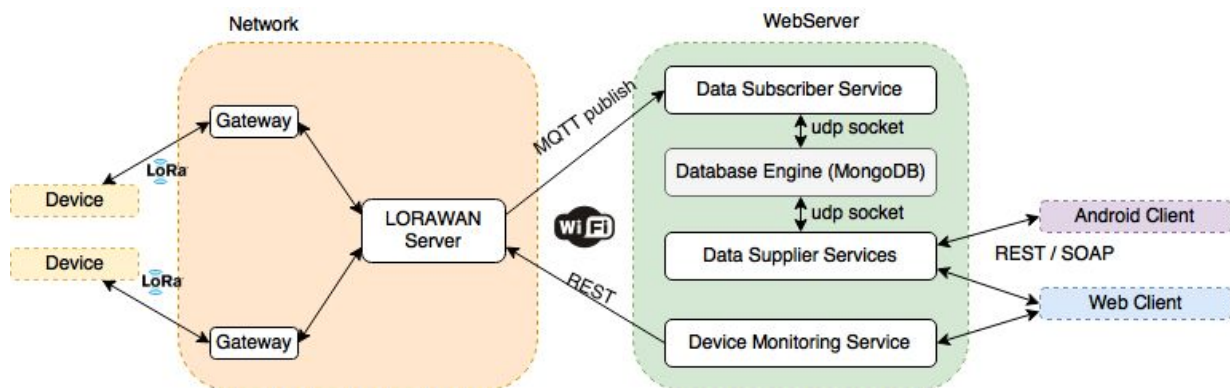


Figure 1: The system components overview

B. Architecture constraints

1. General constraints

We do not have much time to implement our solution as there are many aspects to consider. Our solution involves integrating many different type of technologies of which we may not have the expertise required when starting the project.

Also, the association that we are working with has a very limited budget so we must ensure that our final solution is within their price range.

Finally, the end users of our application will not be aware of the complexity of our system. This means that they might not deploy everything as they should.

2. Network constraints

There is currently no network installation around the Lake Montbel area. No ADSL internet connectivity, no 2G / 3G connectivity or any other kind of network. We must operate our network locally.

3. Web Server constraints

As there is no internet connection, the web server can not rely on any external service (Google Maps, Bootstrap CDN, Angular CDN, etc...). All the static files should be served by the server. Any CDN usage is not possible.

4. Web Application constraints

The Web Application should work on Android, iOS, Windows, Linux, macOS. Supported browsers include: Chromium/Google Chrome, Mozilla Firefox, Microsoft Internet Explorer 10 or later, Safari, Webkit.

The application should behave correctly for both mouse and touchscreen and for desktop screen size as well as mobile screen sizes.

5. Android Application constraints

The Android application should not rely on any internet connection. It should not rely on a connection with the web server while operated on a boat (retrieving buoys GPS coordinates); however a connection to the web server is supposed existing when exporting the data to the server.

The Android application should be compatible with terminals equipped with Android 4.4 or more recent versions. Any hardware using an older version of Android might not be able to use our application. However, only a very small minority of people still use a version of Android older than 4.4 Kit Kat.

6. Device constraints

The devices will be operating in an environment that may be exposed to water and harsh conditions. Therefore, our equipment should be at least rated IP66.

The area that we will be covering is very big. To be able to cover such an area, we may need to use equipments that consume more energy.

We also need to consider public safety because using an antenna that emits with a lot of power might be dangerous. Moreover, a more powerful antenna will be expensive. For this reason, we must find the right balance for our solution.

Finally, our GPS module has to be out in the open to be able to receive a good signal otherwise it would cease to work appropriately.

C. Architecture interfaces

1. Device to Gateway

The devices send short messages to the gateways through the LoRa network.

The message composition is as follows :

Message Field	Size (bits)	Description
latitude	10	The GPS latitude of the device, relative to the race area. A value of 0 indicates the southern position and 1023 (maximum value) indicates the northern position.
longitude	10	The GPS longitude of the device, relative to the race area. A value of 0 indicates the western position and 1023 (maximum value) indicates the eastern position.
Battery level	4	The battery level of the device on a 4 bit scale.

The following figure illustrates the coordinate mapping system (from 32-bit floats to 10-bit integers):

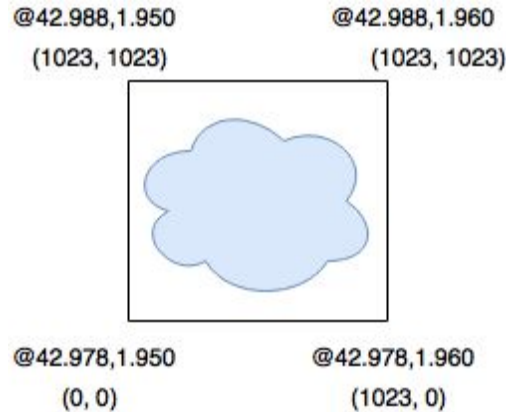


Figure 2 : Illustration of the coordinate mapping system.

The approximation used is valid for small distances only, and is less and less valid as we move to the poles.

Nota : As the lake surface is around 3x3 km, we can represent distances of down to 3 meters in the lake area using 10 bit integers.

2. Gateway to LoRaWAN

a) System schematic and definitions

The protocol between the gateway and the server is purposefully very basic and to be used for demonstration purpose, or on private and reliable networks. There is no authentication of the gateway or the server, and the acknowledges are only used for network quality assessment, not to correct UDP datagrams losses (no retries).

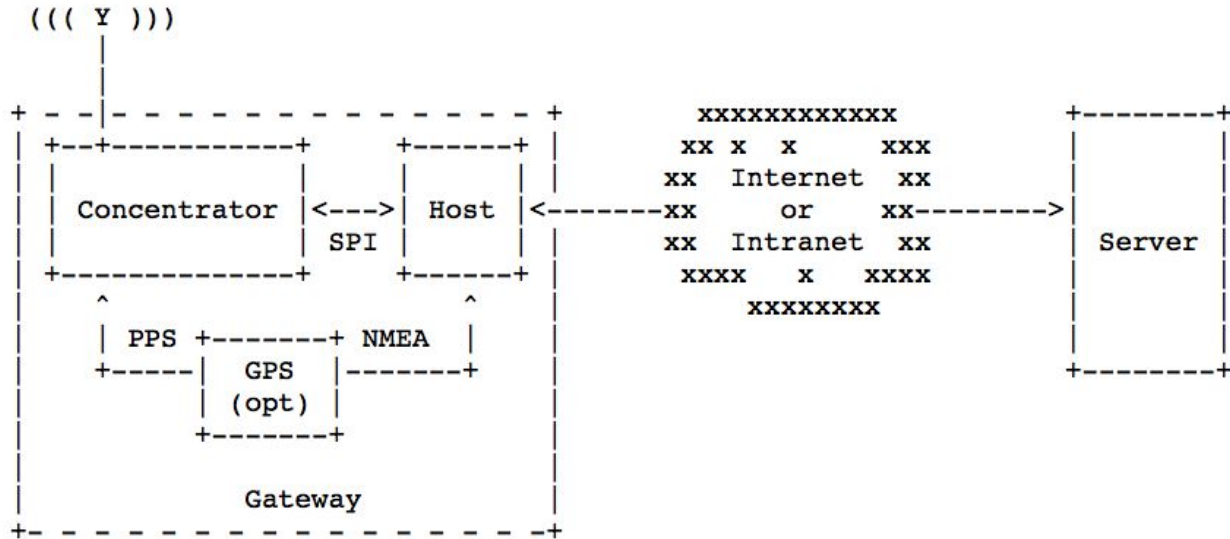


Figure 3 : System schematic

Definitions of these terms:

- **Concentrator:** Radio RX/TX board, based on Semtech multichannel modems (SX130x), transceivers (SX135x) and/or low-power stand-alone modems (SX127x).
- **Host:** An embedded computer on which the packet forwarder is run. Drives the concentrator through a SPI link.
- **GPS:** GNSS (GPS, Galileo, GLONASS, etc) receiver with a "1 Pulse Per Second" output and a serial link to the host to send NMEA frames containing time and geographical coordinates data.
- **Gateway:** A device composed of at least one radio concentrator, a host, some network connection to the internet or a private network (Ethernet, 3G, Wifi, microwave link), and optionally a GPS receiver for synchronization.
- **Server:** An abstract computer that will process the RF packets that are received and forwarded by the gateway, and issue RF packets in response that the gateway will have to emit. It is assumed that the gateway can be behind a NAT or a firewall stopping any incoming connection. It is

assumed that the server has a static IP address (or an address solvable through a DNS service) and that it is able to receive incoming connections on a specific port.

b) Upstream protocol

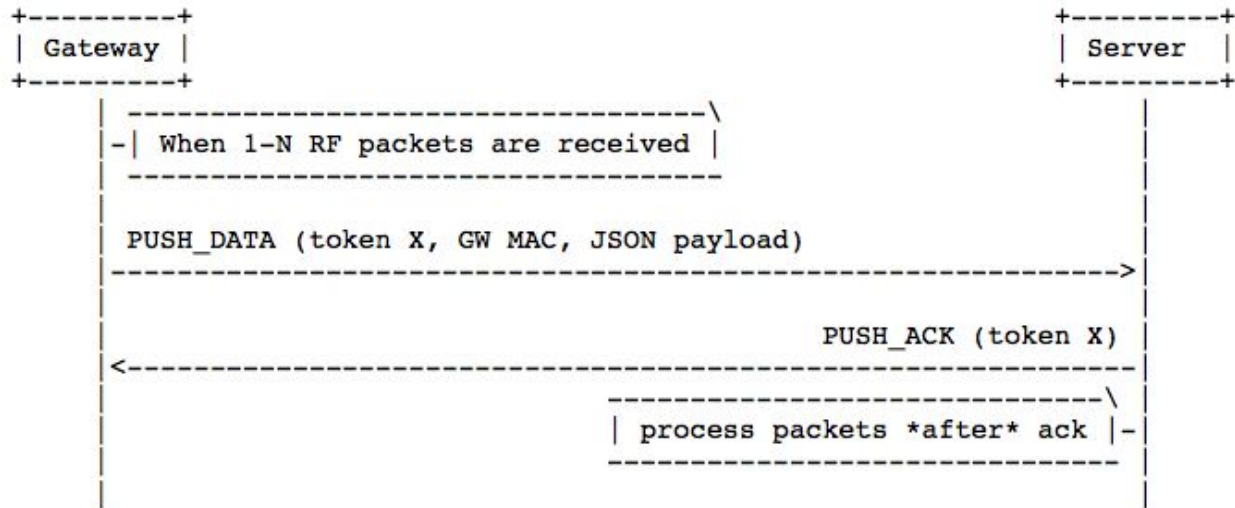


Figure 4 : Upstream protocol sequence diagram

`PUSH_DATA`: That packet type is used by the gateway mainly to forward the RF packets received, and associated metadata, to the server.

Bytes	Function
0	protocol version = 2
1-2	random token
3	<code>PUSH_DATA</code> identifier 0x00
4-11	Gateway unique identifier (MAC address)
12-end	JSON object, starting with {, ending with }

Upstream JSON data structure : The root object can contain an array named "rxpk":

```

{
  "rxpk": [ { ... }, ... ]
}
  
```

That array contains at least one JSON object. Each object contains a RF packet and associated metadata with the following fields:

Name	Type	Function
time	string	UTC time of pkt RX, us precision, ISO 8601 'compact' format
tmst	number	Internal timestamp of "RX finished" event (32b unsigned)
freq	number	RX central frequency in MHz (unsigned float, Hz precision)
chan	number	Concentrator "IF" channel used for RX (unsigned integer)
rfch	number	Concentrator "RF chain" used for RX (unsigned integer)
stat	number	CRC status: 1 = OK, -1 = fail, 0 = no CRC
modu	string	Modulation identifier "LORA" or "FSK"
datr	string	LoRa data rate identifier (eg. SF12BW500)
datr	number	FSK data rate (unsigned, in bits per second)
codr	string	LoRa ECC coding rate identifier
rssi	number	RSSI in dBm (signed integer, 1 dB precision)
lsnr	number	Lora SNR ratio in dB (signed float, 0.1 dB precision)
size	number	RF packet payload size in bytes (unsigned integer)
data	string	Base64 encoded RF packet payload, padded

The root object can also contain an object named "stat" :

```
{
  "rxpk":[ {...}, ...],
  "Stat":{...}
}
```

It is possible for a packet to contain no "rxpk" array but a "stat" object.

```
{
  "stat":{...}
}
```

That object contains the status of the gateway, with the following fields:

Name	Type	Function
time	string	UTC 'system' time of the gateway, ISO 8601 'extended' format
lati	number	GPS latitude of the gateway in degree (float, N is +)
long	number	GPS longitude of the gateway in degree (float, E is +)
alti	number	GPS altitude of the gateway in meter RX (integer)
rxnb	number	Number of radio packets received (unsigned integer)
rxok	number	Number of radio packets received with a valid PHY CRC
rxfw	number	Number of radio packets forwarded (unsigned integer)
ackr	number	Percentage of upstream datagrams that were acknowledged
dwnb	number	Number of downlink datagrams received (unsigned integer)
txnb	number	Number of packets emitted (unsigned integer)

3. LoRaWAN to Data Subscriber Service

The LoRaWAN server uses the MQTT protocol to publish data retrieved from the devices.

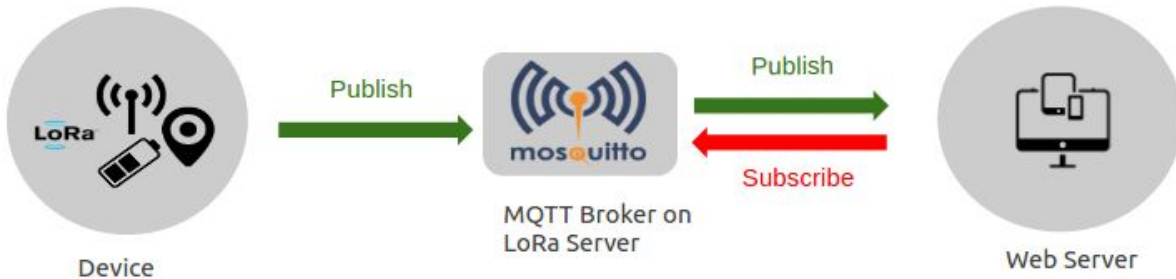


Figure 4: MQTT publish/subscribe messaging protocol.

The LoRa server is installed with an MQTT broker in order to enable communication between the device and our web application. First, the Lora gateway bridge subscribes to the gateways topics. When the device publishes data on those topics, the gateway bridge will be notified and proceed by transmitting this data to the Lora server. Our web server application has also an MQTT client that can subscribe those topics received by the Lora server. The next figure gives a general overview:

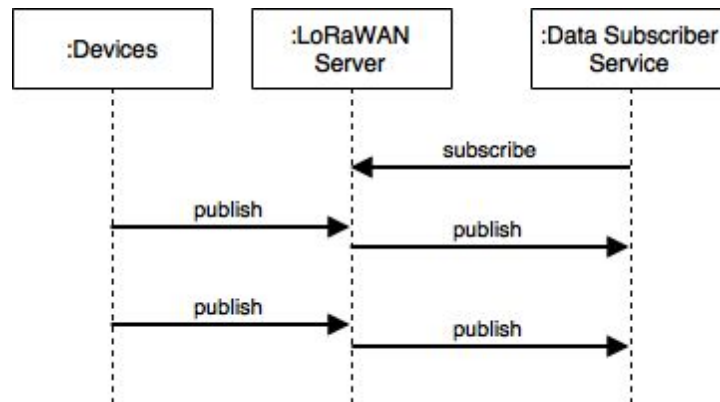


Figure 5: MQTT protocol sequence diagram.

For example, to receive everything sent by a device to our gateway with ID b827ebffffa9d226, we need to subscribe to this gateway with the following command:

```
mosquitto_sub -t "gateway/b827ebffffa9d226/#" -v
```

This will display every data from the gateway. Then our web application, that has also already subscribed to the same gateway, will receive the data to exploit them and retrieve the GPS coordinates.

4. Services to MongoDB

The connection between services and the database backend is made using a classical MongoDB connector.

5. Data Provider Services to Android and Web Clients

The Data Provider Services (**DPS**) expose their data through REST and SOAP APIs.

Each DPS exposes a subset of the database. Each service is a separate entity because some services might be replaced by new services that may retrieve their data from other third-party software instead of the MongoDB backend.

For instance, the race participants could be retrieved from another software, like the one CVRL uses to registrate competitors.

The following table enumerates the raw data services, used to access database data.

Service Name	Technology	Description
RaceDataService	Node.js REST/JSON	Serves the raw race data : the list of successive GPS position records for each racer.
RegataService	Node.js REST/JSON	Serves regata data : list of races, location, start and end dates.
RaceService	Node.js REST/JSON Java SOAP/XML	Serves race metadata such as date, list of racers, race map used, buoys positions etc...
RaceMapService	Node.js REST/JSON	Serves race map images and localization data of the race area.
DeviceService	Node.js REST/JSON	Serves device data such as Hardware Device Identifier, Short Name, and device metadata such as battery level.

The following table enumerates the composite services, built upon the raw data services :

Administration Service	Node.js REST/JSON	Composite service that uses simpler services to perform administration tasks.
RaceStreaming Service	Node.js REST/JSON	Composite service that can be used to stream races.

BPELService	Java XML/SOAP	Composite service used by the Android application which manages the buoys.
-------------	---------------	--

6. Device Monitoring Service to LoRaWAN server.

To send packets to the devices, the Device Monitoring Service must send these packets to the LoRaWAN server downlink queue. It uses the built-in REST of the LoRaWAN server.



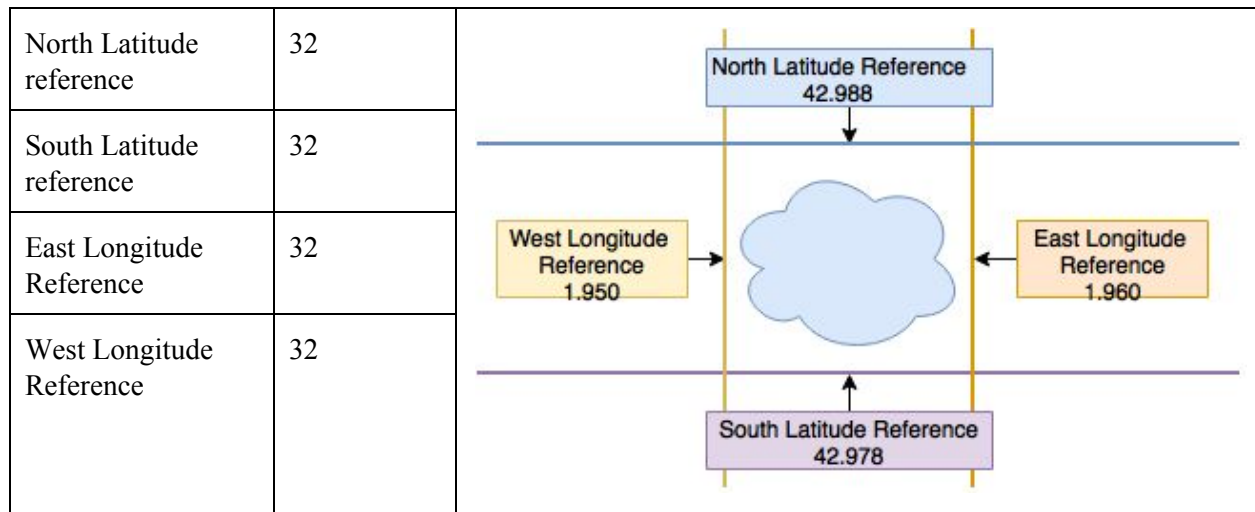
Figure 6: Downlink communication.

The Device Monitoring Service can send the following messages to the LoRaWAN server, which will broadcast them to the devices :

Message Field	Size (bits)	Description
Operation Code	2	The operation code indicates the type of message the device will receive. 0 : sleep mode activation. The device will only send 1 time each 30 minutes. 1 : tracking mode activation. The device will enter normal operation mode. 2 : Set geographic reference. This is used to set the geographical coordinates of the area of the lake.
Operation Data	128	Only used by the “Set geographic reference operation”.

In the case of a “Set geographic reference” message, the payload is structured as follows :

Operation Data field	Size (bits)	Description
----------------------	-------------	-------------



D. User interfaces

1. Web Application

Our application only implements text inputs. Several views require text input, such as the edition of regattas, races, devices etc, and the authentication of the user.

2. Android Application

The person in charge of installing the buoys on the lake is the only user of the application.

The application will have two different task available regarding where it is located. When the user is on the lake installing the buoys, he will have the possibility to press a button each time that he has placed one buoy to make a record of the location. When he gets back to the base area, he can proceed to transfer the data containing the positions of all the buoys to a given database.

II. Detailed component design

A. Web server

The Web server is, in fact, composed of two servers built using different technologies.

The first one is the Node.js server that will host all the services described in “Architecture Interfaces - Data Provider Services to Android and Web Clients” section.

The second one is the Glassfish server hosting java web services and BPEL. The Glassfish server will be used for **learning purposes** only, it won't be shipped in the final delivery. The services offered by the glassfish server will also be implemented as Node.js services, in order to avoid shipping a Java 6 / Glassfish environment to the client.

1. Node.js server

a) Database schemas

As the database engine is MongoDB, the database is organized in collections containing documents. The class diagram below summarizes the data present in the database and manipulated by the web server.

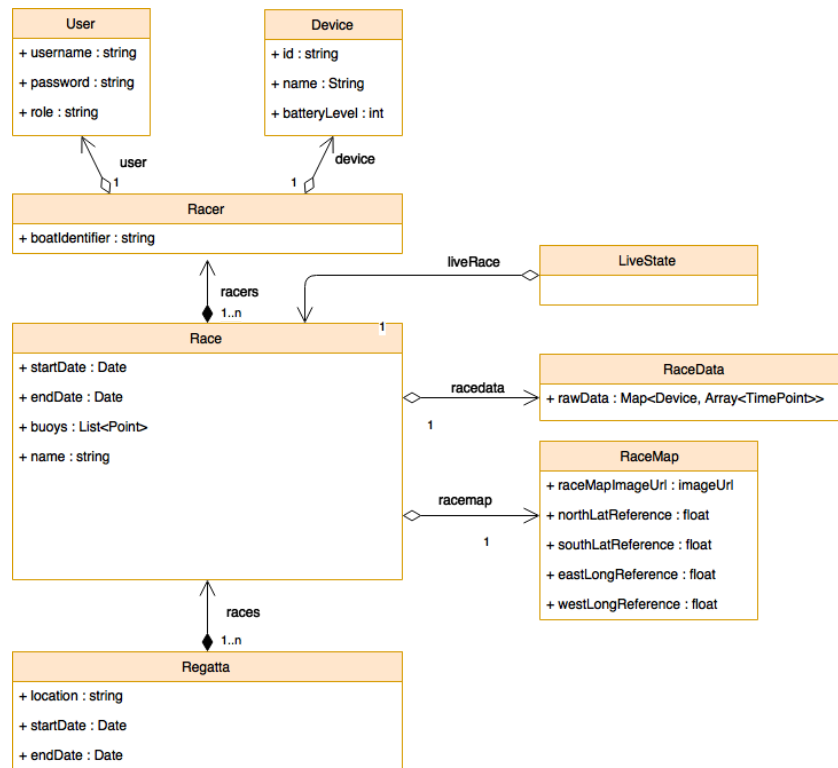


Figure 7: The system's class diagram.

Below is a list of all the collections and the document schemas that are contained in these collections.

Users : represents the system users.

Field name	Field Type	Description
username	string	Username of the user.
password	string	Password of the user. In a future version, this field might only contain a hash of the password for security purposes.
role	string	Role of the user. “member” : Association Member “staff” : Association Staff Member

RaceMap : represents a race map that can be reused along many races. As the location of the buoys may vary from race to race, the race map does not store the buoys position.

Field name	Field Type	Description
name	string	Name of the race map.
raceMapImageUrl	string	Url of the race map image.
northLatReference	float	Northern latitude reference of the area represented by the race map image.
southLatReference	float	Southern latitude reference of the area represented by the race map image.
eastLongReference	float	Eastern latitude reference of the area represented by the race map image.
westLongReference	float	Western latitude reference of the area represented by the race map image.

Regatta : represents a competition containing a set of races.

Field name	Field Type	Description
races	List<Race>	List of references to the races of this regatta.
location	string	Location of the regatta.

startDate	Date	Start date of the regatta.
endDate	Date	End date of the regatta.

Race : represents a race.

Field name	Field Type	Description
map	Ref<RaceMap>	Reference to the map of this race.
data	Ref<RaceData>	Reference to the raw race data corresponding to the data retrieved during this race.
buoys	List<Point>	List of GPS position of the buoys. Nota : Points are { x: 0, y: 0 } data structures.
racers	List<Racer>	List of the racers participating in the race.
startDate	Date	Start date of the race.
endDate	Date	End date of the race.

Racer : a racer represents the association of a boat identifier, a device and (optionally) a user registered in the IoTracking system. There are no collections containing racers, they are contained in the **racers** field of the **Race** documents.

Field name	Field Type	Description
boatIdentifier	string	Unique immatriculation of the boat.
user	Ref<User>	Reference to the user which is considered as the skipper of the boat. This value can be null.
device	Ref<Device>	Reference to the device which was used to track the boat during the race.

RaceData : raw GPS tracking data retrieved during a race.

Field name	Field Type	Description
rawData	map<Ref<Device>, List<TimePoint>>	Map that makes the correspondence of each device with a list of time points. Time points are objects with the following fields :

		t	int	UNIX Timestamp corresponding to the time where the gateway received the message.
		x	int	Longitude retrieved from the device (scale 0-1023)
		y	int	Latitude retrieved from the device (scale 0-1023)

Devices : these documents represents the real word tracking devices with their unique Hardware Identifier and their name (the name which can be printed on the device to easily recognize it, for instance DEVICE-0023).

Field name	Field Type	Description
identifier	string	Unique Hardware Identifier of the device.
name	string	Name of the device.
batteryLevel	int	Last retrieved battery level of the device.
lastActivity	Date	Date representing the last time the device emitted data.

LiveState: represents the state of the live session. Only one live session is supported.

Field name	Field Type	Description
liveRegata	Ref<Regata>	Identifier of the regatta which holds the live race.
liveRaceId	int	Identifier of the live race. (index of the race in the regatta races array).

b) Auto-generated raw data services

As we want our system to be flexible, we built it in a way that all the collections could be accessed directly through a REST API. We created a REST API generation framework from our schemas, which automatically generates the GET/POST/PUT/DELETE apis and validates the input given for POST/PUT requests. Here are the HTTP methods available for all the collections :

Method	URI	Content Type	Possible Answers
POST	<i>/ {collection}</i>	application/json	201 : created 400 : bad object format or property with incorrect value.
PUT	<i>/ {collection} / {id}</i>	application/json	200 : OK 400 : bad object format or property with incorrect value. 404 : no object with the given id.
GET	<i>/ {collection}</i>	NA	200 : OK. Lists all the documents of the collection as json.
GET	<i>/ {collection} / {id}</i>	NA	200 : OK. Returns the document with the given identifier as a json object. 404 : no object with the given id.
DELETE	<i>/ {collection} / {id}</i>	NA	200 : OK. 404 : no object with the given id.

c) Other services

Buoy Service : this service is built to replace the list of buoys of a race with a simple API call, without updating a full regata. It is designed to work with the Android application.

Method	URI	Possible Answers	Description
PUT	<i>/api/buoys/ {regata} / {race}</i> Where : - regata : id of the regata. - race : index of the race in the races of the regata.	200 : OK 400 : bad object format 404 : regata not found, or race not found	Set the buoys field of the race identified by the couple ({regata}, {race}).

Live Service : this service groups functionalities related to the live race broadcast functionality.

Method	URI	Possible Answers	Description
GET	/api/state/live	200 : OK 400 : bad object format 404 : regata not found, or race not found	If OK, returns : - {} if there is no live race. - A JSON formatted LiveState object is there is a live race.
DELETE	/api/state/live	200 : OK	Terminates the live.
POST	/api/state/live	200 : OK 400 : bad object format.	Sets the live state. Takes as argument : a JSON formatted LiveState object..

d) Composite services

Here is the list of composite services :

Buoy Service	Service that simplifies the insertion of buoys into a race.
Data Subscriber service (MQTT client)	Service that subscribes to notifications of the LoRa server about the data sent by the devices, and that invokes other services to add tracking data to the racer who holds the device in the active race.

Buoy service

Method	URI	Possible answers	Description
PUT	api/buoys/{regata}/{race}	200 : OK 400 : bad object format 404 : regata or race not found.	Takes as argument : a Point[] object representing a list of absolute GPS coordinates. Sets the buoys coordinates (absolute) of the given race of the given regata.

Data Subscriber service (MQTT client)

This service is not an HTTP service, it is a service that listen for MQTT broadcast messages on the LoRa Server.

Listens on : {loraserver_ip}

Subscribes to topic : application/#

Incoming Message format : see section C.1.

The service receives the incoming messages, and extracts the following information from the message :

Incoming message format.

Field	Type	Description
time	Date	Date of reception of the message.
devEUI	string	Device identifier.
content	MessageContent	Message content. The message content is extracted from a base64 encoded binary string.

Reminder : MessageContent format.

Field	Type	Description
x (longitude)	integer (10 bits)	Longitude component of the coordinates.
y (latitude)	integer (10 bits)	Latitude component of the coordinates.
batteryLevel	Integer (4 bits)	Battery level of the device.

2. Glassfish server

Our system provides a service to get information about the upcoming regattas and the racers that participate in these regattas. Indeed, our client uses an application provided by the French Sailing Federation. This software is named Freg and allows the user to enter several types of information about a regatta. In order to avoid them to re-enter this data in our application, we provide a parsing service that imports a CSV file (exported from Freg), parses it, and set the data in our database.

We use SOAP web services, orchestrated by a BPEL composite application to implement this service. It is implemented into three services:

Our first importCSV web service imports the CSV file as a string and transmits it to the second service.

This second web service, parseCSV, parses this string and extracts the relevant data we need, to build a new string.

Finally, this string is sent using a REST request to our third service, which is an external bus service, on our application server. ImportRacers also send an acknowledgement to our first service.

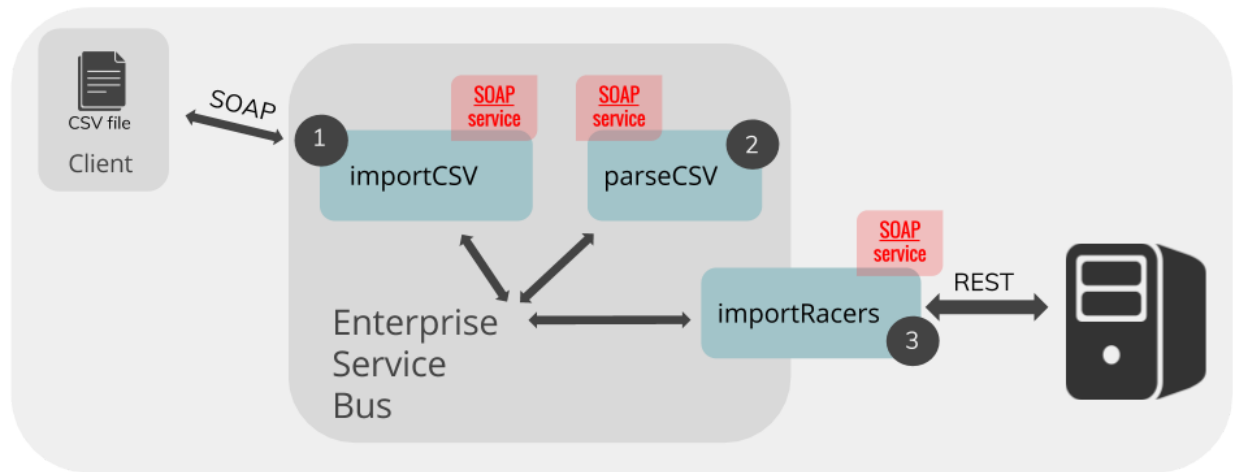


Figure 8: Schema of our ESB architecture.

1. ImportCSV

input	string	Acknowledgement from ImportRacers
output	string	CSV file as a string

2. ParseCSV

input	string	CSV file as a string
output	string	Parsed CSV file as a string

3. ImportRacers

input	string	Parsed CSV file as a string
output	string	Acknowledgement to ImportCSV as a string

4. ExportRacers

output	string	Parsed CSV file as a string
--------	--------	-----------------------------

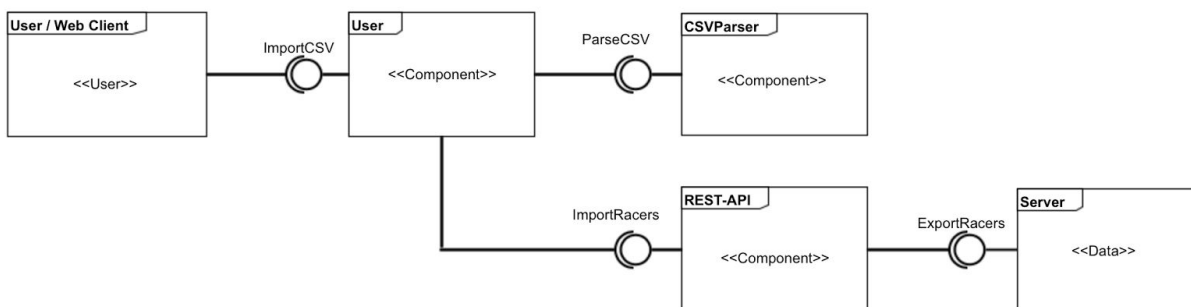


Figure 9: Component diagram of the ESB architecture.

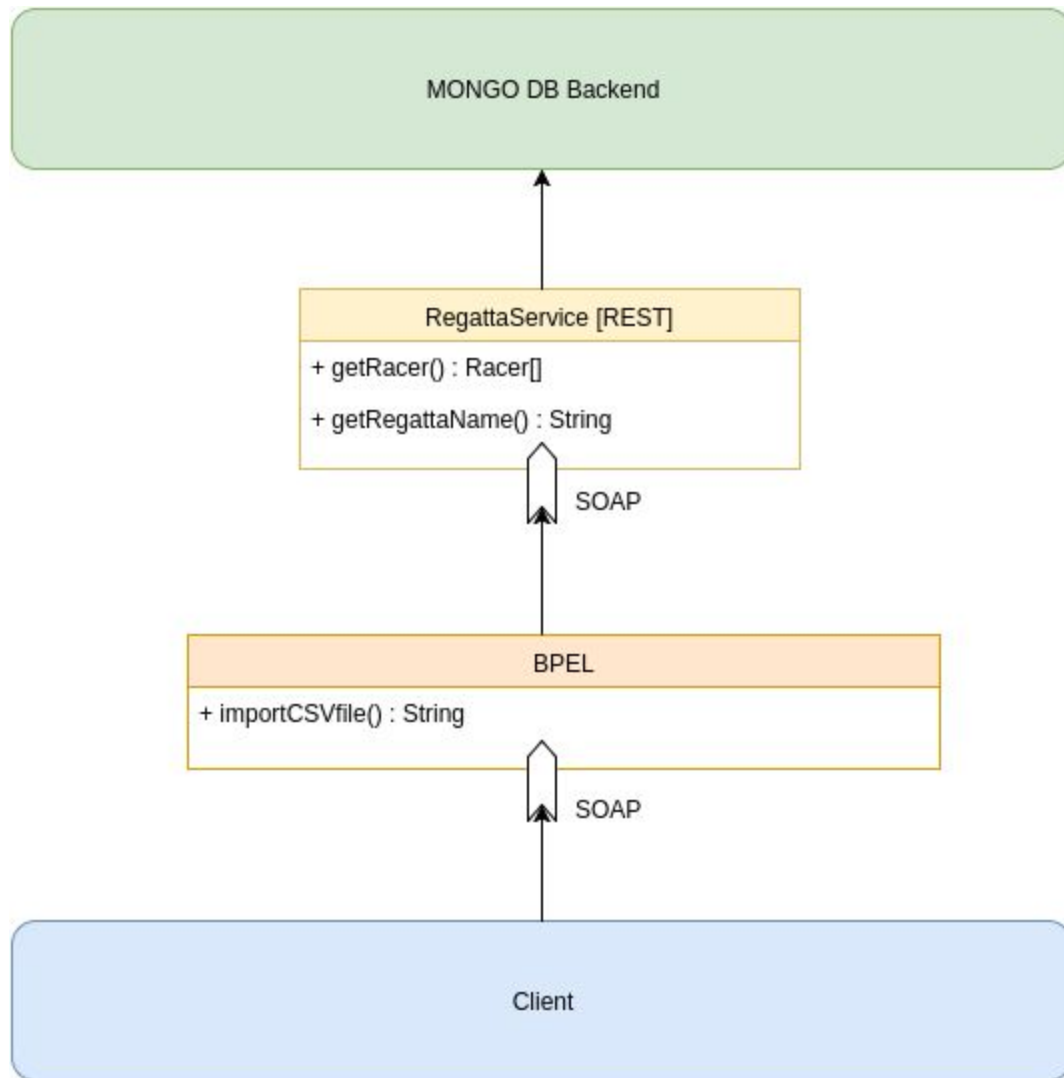


Figure 10: Schema of our ESB architecture.

B. Web client

1. Architecture overview

The web client is an HTML / Typescript based client, that can be served through a simple HTTP server unlike JSP or ASP pages. It communicates with the web services through AJAX queries.

We are using the Angular 2 framework, and following the Component Oriented Design guidelines. The following figure illustrates the web client architecture :

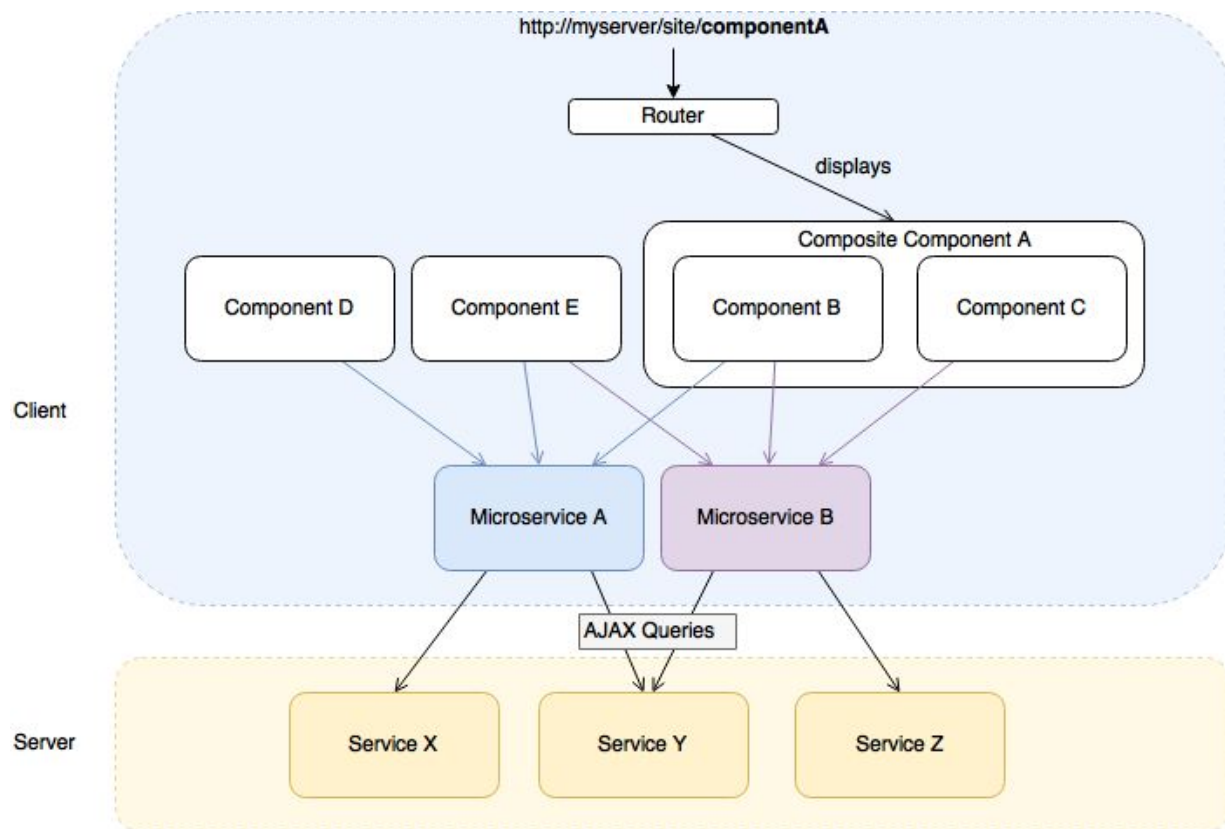


Figure 11: Overview of the client architecture.

Whichever URL the browser asks for under <http://myserver/site/>, it is served the same index.html file. However, the angular framework manipulates the dom in a way that it can display components depending on the route the browser asks for. For instance, if the browser asks for <http://myserver/componentA>, we can configure the Angular router so that it displays a specific component (Composite Component A for instance in the figure).

Considering this, the application is subdivided in components. Some components will be served by the router while others will be used by other components, called composite components.

Components can hold client specific state, and are associated a HTML template. They get data from the server by communicating locally with microservices that can be shared across multiple components.

These microservices communicate asynchronously with the web server through AJAX Queries.

2. Microservices

In this section, we will enumerate the main microservices and their roles.

UserService : this service is responsible for managing user authentication. It also saves the authentication state (username and role) in order to display or not some features those access is restricted to a certain type of user. If some feature requires specific authorization on server-side, they will not be shown on the client.

RegattaService : provides data and functions for manipulating regattas and races.

DeviceService : provides data and functions for manipulating devices.

3. Main components and routes

In this section, we are not going to describe all the components of the application, but instead focus on the routes we are using and the main components to which they correspond.

a) Staff member and administration views

Dashboard : This view provides access to administration panels of the regattas, race maps and devices.

Regattas : This view enables the management of the regattas : creation, deletion, edition of upcoming and past regattas.

Races Edition : This view enables the editing of a race : start and end dates, map used, buoys position viewing (they are set using the Android Application), race tracking data, list of racers. The list of racers can be imported from CSV files coming from the freg software.

Devices : This views helps the staff members to keep track of their devices.

The staff members can change the name of the devices, whose hardware identifiers are unique. The new devices (which do not yet have a name) are shown to the user, so they can register them. The devices that have a low battery level are also shown, as well as the active devices.

Race maps : This view enables the edition of race maps : geographical references and background image.

b) Anonymous user views

Home : the landing page of the anonymous users. Users can access to the other views from this view.

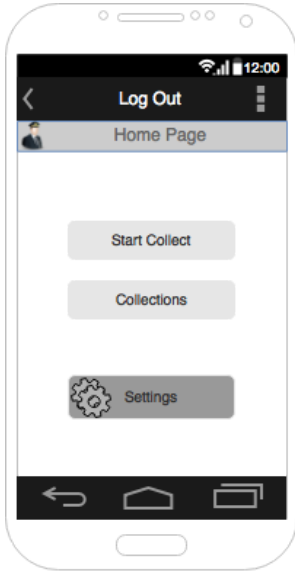
Live Race : view where the race occurring live is displayed, if there is one.

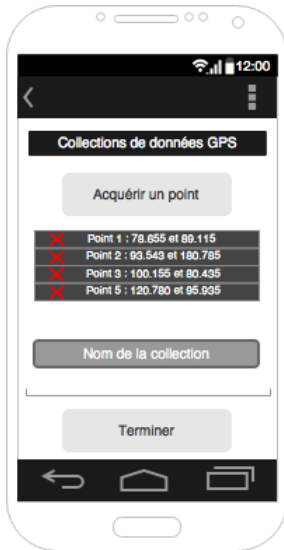
Past Regattas : view where users can view past regattas and have access to the Deferred Race view.

Deferred Race : view where users can replay past races.


C. Android Client


1. Use case


GUI	
Objective	<p>This page allows the user to do three things:</p> <ul style="list-style-type: none">• Start acquiring GPS coordinates: Start Collect• See previously saved collections: Collections• See the the different servers available: Settings

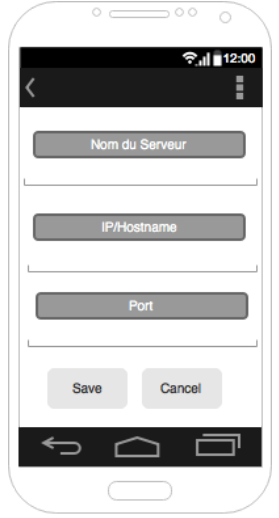
GUI	
Objective	<p>This page allows the user to do three things:</p> <ul style="list-style-type: none">• Acquire GPS coordinates: Acquérir un point

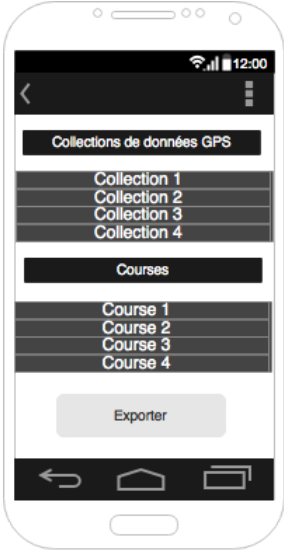
	<ul style="list-style-type: none"> Type out the desired name for the collection Save the collection: Terminer
--	---

GUI	
Objective	<p>This page allows the user to do three things:</p> <ul style="list-style-type: none"> Select a collection Modify a collection or see the coordinates: Modifier la collection Start acquiring new GPS coordinates: Ajouter une collection

GUI	
Objective	<p>This page allows the user to do two things:</p> <ul style="list-style-type: none"> Delete the collection: Supprimer la collection Go back to the home page: Annuler/Terminer

GUI	
Objective	<p>This page allows the user to do two things:</p> <ul style="list-style-type: none"> • Select a server • Add a new server: Ajouter un serveur

GUI	
Objective	<p>This page allows the user to do three things:</p> <ul style="list-style-type: none"> • Type in all the required fields • Save the changes: Save • Cancel the changes: Cancel

GUI	
Objective	<p>This page allows the user to do two things:</p> <ul style="list-style-type: none"> • Select a collection • Select a course • Export the selected collection to the server: Exporter

D. Network

To connect the gateways to the network server, we will need to deploy a local WiFi network. To achieve this, we will require WiFi antennas that can help us cover the whole area of the lake. As we will have 3 gateways, we will need 3 directional antennas for each gateway. These antennas will be facing back to the nautical base where an omnidirectional antenna will be deployed.

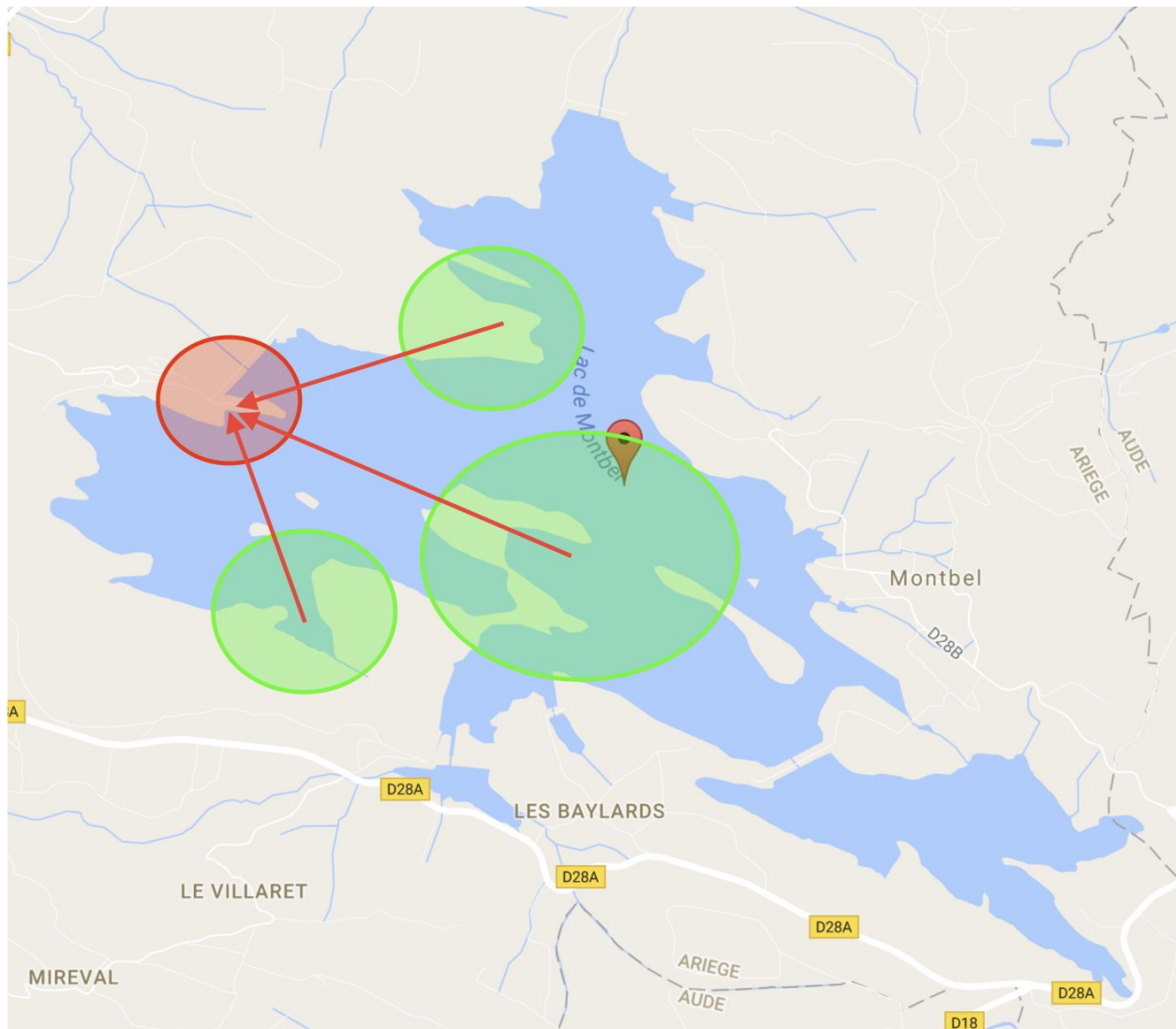


Figure 12: Network triangularisation.

Based on where the gateways will be placed, the location of the antennas will be similar to the figure above. The green circles represent where a directional antenna will be placed and the red circle represents where the omnidirectional antenna will be placed. Ideally, the horizontal transmission angle of the antenna at the nautical base should be at least 120° facing the other three antennas. The horizontal

transmission angle of the other 3 directional antennas should be at most 30°. The antennas should be powerful enough to transmit over a distance of 2km.

E. Device

The device is a LoRaOne card that includes:

- An Arduino compatible microcontroller: ATSAMD21G18, 32-Bit ARM Cortex M0+
- A LoRa module: Microchip RN2483 Module
- A GPS module: uBlox EVA 7M
- An accelerometer/magnetometer: LSM303D

The device has two different modes, sleep mode and active mode. This corresponds to an off/on mode, and eliminates the need of a physical switch to turn the device on and off.

The accelerometer have been configured so that it generates an interrupt when it detects a movement in either x- or y-axis. This interrupt is used to wake-up the device so that it passes from sleep mode to active mode.

The device has a function that measures the current battery level of the power source.

After having retrieved the coordinates via the GPS module, the device calculates a set of relative coordinates according to a set of reference coordinates.

Every tenth second, the device sends a packet of 3 bytes via the LoRa module. This packet contains the relative GPS coordinates and the battery level of the device.

At the end of each while loop the device will enter sleep mode, but it will immediately wake up again when in movement. When in storage between usage, the sleep mode will assure a minimal power consumption of the device.

III. System deployment

The system consists of hardware components, network components, and software components. In this section we explain how we plan to deploy the components.

A. Software configuration and deployment

1. Deployment constraints

As CVRL doesn't have any internet connection at the lake, we cannot and will not deploy our software on a remote cloud.

We want the price of the hardware needed to run our software to be low. As a consequence, our software must be able to run on a low end server (requirements : 6Go RAM, 50Go HDD, dual-core CPU).

Also, we want our software to run alongside other potential applications on the association server.

Finally, we want our software to be easily deployable. However, it is required to have an internet connection during deployment.

2. Deployment overview

Considering the above constraints, we will not deploy a IaaS platform (such as OpenStack), as it requires running more nodes, which means more RAM.

However, we still want to virtualize our software, so they can run in isolated environments, and will let the host operating system clean.

Considering the amount of RAM available, we also want to minimize the amount of virtual machines running on the host operating system.

Figure 6 shows an overview of how the software components will be deployed.

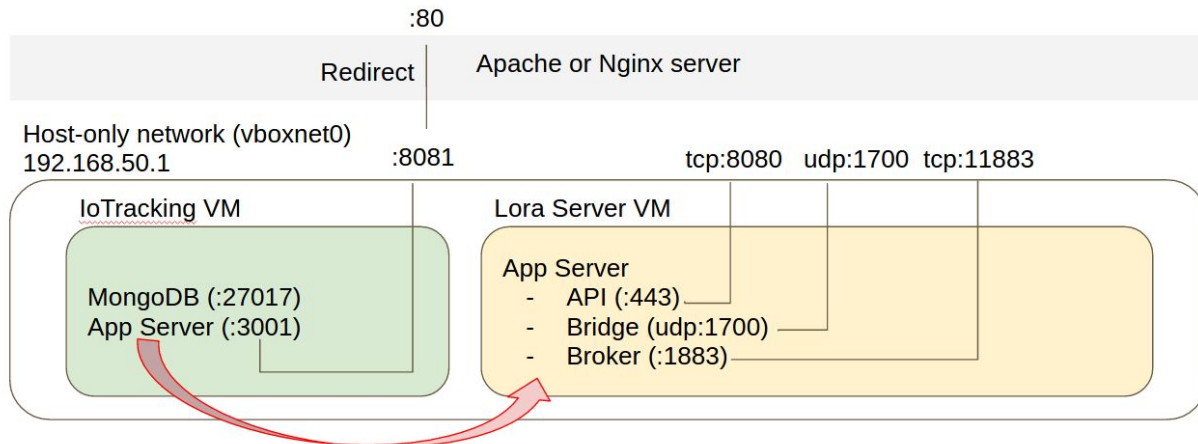


Figure 13: Illustration of the deployment schema.

3. Software configuration

a. Apache - Nginx Server

This server will intercept HTTP requests from the clients, and transfer them to the appropriate handler, based on URL. The choice of the URL is let to the system administrator of CVRL.

Note : if the URL is different from the root (/), appropriate configuration must be done at the level of the Web Client application for proper routing.

The setup shown here implies a redirect of HTTP connections from :80 to port :8081.

b. IoTracking VM

Components

The IoTracking VM will host two components :

- The Application Server : the server running all the services, and serving the Web Client files.
- The MongoDB database.

Although the database could be hosted on a separate virtual machine, we chose to group it with the application server to avoid creating another virtual machine (and thus limiting RAM usage).

Network interfaces

The IoTracking VM has two network interfaces :

Type	Role	
NAT	Port-forwarding.	
	Host port	Guest port
	tcp:8081	tcp:3001
Host-only Network	Access to the host in order to send requests to the LoRa Server VM guest.	

The database is not accessible from the host.

The Application Server is accessible from the host using the 8081 port.

Configuration

The following configuration variables must be configured on the Application Server :

Variable	Value	Example
loraMqttBrokerUrl	mqtt://{host_ip}:{broker_port}	mqtt://192.168.50.1:11883
loraServerUrl	https://{host_ip}:{server_port}	http://192.168.50.1:8080

c. LoRa Server VM

Network interfaces

The LoRa Server VM has only one network interface :

Type	Role	
NAT	Port-forwarding.	
	Host port	Guest port
	tcp:8080	tcp:443
	tcp:11883	tcp:1883
	udp:1700	udp:1700

IV. Annexes

A. Traceability Matrix

This traceability matrix below presents the different requirements of buoys GPS positions system, its current status, and how we implemented it.

REQUIREMENTS TRACEABILITY MATRIX				
Project Name:	IoTracking			
National Center:	INSA			
Project Manager Name:	Cyril ANAK STELL			
Project Description:	GPS positioning for boat regattas in sites with no network coverage			
ID	Assoc ID	Functional Requirement	Status	Implemented In
001	1.1.1	The Android app must make a REST request to get the available races from the database	Completed	Java, REST
002	2.2.2	The Android app must display the races availables for buoys setting and allow the user to choose one of them	Completed	Java
003	3.3.3	The position of the buoys must be get for GPS network when the user clicks on a button	Completed	Java
004	4.4.4	These GPS positions must be recorded in the smartphone	Completed	Java
005	5.5.5	These GPS positions must be send to the server as soon as the user click on "Finish" and there is a WiFi connection	Completed	Java, REST
006	6.6.6	The server must give these positions to the MongoDB database using REST	Completed	Java, REST

Figure 14: Traceability matrix for the Android application development.