***Cloriko***

***Introducction***

Cloriko is the definition of a Cloud architecture made it personal.

*‘And, what does it means?’*

Well, it means that the user is the one responsible of setting up their own cloud.

*‘So, what does even that implies?’*

This can sound like a very difficult process, but in fact it is not, and it is easily defined in only two steps.

* Creating a user account to Cloriko’s web.
* Installing the its software in each of the devices that it is wanted by the user to be part of the Cloriko’s device network. (And logging in)

From here on, the user does have available the access to the documents stored in each of these devices by using the Web App. And the truly power of that is that is the user the one responsible of adding more storage to its ‘cloud’ by just including more devices to the network.

This paradigm has a lot of features yet to discover and be developed, but the most important one is that users will be the owners of their data! Which implies advantages such as having fully privacy or not being managed or asked to pay for any external service/storage.

We, at Cloriko belive that this model can fit in a different range of scenarios, from the single user to the local business and even bigger ones that might need it for managing more sensible data.

***Technical overview***

This section includes the main features of the product:

Make the desired data available from each of the network’s devices and make it accessible from the Web App.

In order to achieve that, different requirements will be needed to be developed:

* Web app that will allow users to register, log in, and see their data.
* Each device is be automatically set up by installing the Cloriko software and logging in it.
* A new folder is being created when installing the software therefore any document can be put there and it will be updated to personal cloud.
* At least one device is be needed to be registered with the possibility of adding multiple devices, thing that would simulate a cloud behaviour.
* Three different storage modalities defined: *Safe* and *Light* and *Duplication*
  + *Safe* : All the content updated to the application is replicated in at least two replicas. Note that it is only possible to use this mode when there have been set at least two devices for the same user and a maximum of 3 recommended.
  + *Lighter* mode does not care about any replicas. So the device’s data would be lost in the case that one of the gets down.
  + *Duplication*: This mode will also be available for those cases in which we want all the data in each device. (This mode is very heavy and it is not really seen as a cloud behaviour, instead it is more seen like a shared folder in which each user has the full copy of the data, allowing with no network latency and accessing the full content in an offline mode (explained below).
* The platform will just be available for online access since it only will be available using to be used from the web app.
* Two privacy modes, *Public* and *Private*.
  + *Public*: Allows users to have the documents their selected opened to everyone (including themselves). So at the end, it avoids the need of having to log in for accessing to any content.
  + *Private*: Just the opposite to Public
* When the data gets deleted, a notification from the device is sent to the Master server whom will handle it in a different way depending on the storage modality currently being used.
  + Safe: In a safe mode the user is asked for choosing between forwarding the deletion

Components design

**Master Node**

The master node can be seen as the kernel of Cloriko. It represents the middle point between any user that wants to access their data and the data itself.

The server is built on top of Akka http and the actor model that communicates to the different slave nodes that composes user’s cluster.

The actor system uses Cassandra database to persist the data related with user’s and the cluster’s metadata.

The following sub sections describes in detail how the Master Node is structured.

**Routes**

*Web app routes:*

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Route** | **Body** | **Description** |
| GET | /home | - | Home page of Cloriko, it shows an introduction to the software. |
| POST | /signUp | { username: String,  name: String,  surname: String,  birth: String,  email: String } | Data sent that has been put by the user and sent by the Web App when clicking to ‘Sing up’ |
| GET | /<username>/share | - | Returns the list of nodes and details that the user has registered |
| GET | /<username>/content/root | - | It returns the root directory for the user to start browsing their content |
| GET | /<username>/content/<path> | - | It returns the overall content found at the path passed as parameter |
| POST | /<username>/copy | { from: String,  to: String,  fileName: String  } | It does a copy of a file. The json defines the origin, destination and file namecopies a file from the specified location to the |
| POST | /<username>/move | { from: String,  to: String,  fileName: String  } | Same as copy but removing origin file. |
| POST | /<username>/delete | { path: String } | It performs the deletion of the selected username |
| POST | /<username>/updated/<path> | Image/jpg/png | It allows the user to update images passed as a body of the request, into the path passed at the route |

**–** *Cluster routes:*

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Route** | **Body** | **Description** |
| POST | /share/join | { username: String,  shareId: String,  id: String,  deviceType: String,  maxCapacity: String} | A new share wants to join to the username cluster. It sends their |
| POST | /share/left | { username: Sting,  shareId: String,  id: String } | The share left the cluster shares sending a leaving message. |
| POST | /share/update | { username: String,  shareId: String,  id: String,  update:  { shareId: String,  name: String,  path: String,  filetype: String,  filesize: String }  } | - |
| POST | /share/updates | { username: String,  shareId: String,  id: String  updates:  [ { filename: String,  path: String,  filetype: String,  filesize: String } ] | Send the information of the file updates |
| POST | /<username>/share/delete | { username: String,  shareId: String,  id: String,  delete:  { filename: String,  path: String }  } | A single file was deleted and its metadata was sent with the request. |
| POST | /<username>/share/deletes | { username: String,  shareId: String,  id: String,  deletes:  [ { filename: String,  path: String } ]  } | Different files were deleted and its information was sent to the master to forward that changes to Cassandra. |
| GET | /<username>/share/renamed | { username: | It returns the root directory for the user to start browsing their content |
|  |  |  |  |

**Slave Node**

**Web App**

**Future work**

* Allow multiple contexts & clusters for the same user
* Subscriber model: A user can be able to subscribe to other user’s content, in that case the data updated by the secondary user does not infer the primary one and neither the secondary user device’s will be seen as part of the primary’s device network.