Bachelor project

# 27.04

I’ve decided to use nodeJS and expressJS for the backend.  
I considered using firebase but unfortunately there’s no way to store data internally and that would be a problem as we’re handling sensitive data from the patients.

They’ll be used to build a RESTfull API to communicate with a MySQL database (see if maybe it would be better to save data directly in HUG temp database).

# 28.04

We’ve talked with Mr. Gluck about the scope of the project and redefined it a little by adding IBeacon detection and location within the HUG mainframe.  
To do that I must use Bluetooth LE technology which is not available by the means of a web app.

We decided to switch to native applications both on android and ios.  
I now have to search and find the best suited framework between ionic phonegap and apache cordova to build my application.

We talked with Mr.Ehrler about my gantt chart and what where the main issues that had to be fixed. Basically I had to add some simple technical description a little more in depth than what I did for each main task.

# 29.04

I’ve decided to use ionic for several reasons.

Phonegap is basically apache cordova with addons for adobe softwares which I have no interest using.

Apache cordova is great and would have worked just fine but I would have had to redo a lot of work and spend time learning their API.

Why Ionic was a better fit and cordova ?  
Ionic is based on angularJS, which is great because it was part of what I learned to use during my semester project so I would be able to save a lot of time using it.

Both have live preview of the in development app through native applications found on the app store and play store.

Both are based on web technologies and have the ability to work with IBeacon.

# 30.04

I’ve finished updating my gantt chart and sent it to Mr.Gluck and Mr.Ehrler.

I’ve also started working on ionic and was able to display the list of interventions ordered by time and type as I did for the semester project.

I’ve decided to use a modular structure for my project because of several advantages it gives for big projects development and code maintaining.

Here’s a list of those advantages:

* Less coupling. Because of the strict separation of logical blocks of code, cross-referencing between features is discouraged. If you’re going to have a feature that depends on another feature, you’ll have to explicitly specify this by having one module require another. It’s a little more work, but you’ll think twice before doing so and it makes the relationship a lot easier to spot.
* More confidence. If someone unfamiliar with the project has to fix a bug, he’ll have less trouble comprehending the implications of his changes since dependencies are easy to spot.
* Less mocking. Writing unit tests becomes a lot easier because the reduced number of dependencies means there’s less stuff you need to mock in your tests. Since each feature is a module, you can test them as if they’re a separate application.
* More code re-use. By grouping code feature-wise, it becomes a no-brainer to copy a whole feature from one project to another since you can simply take the whole directory. With the folders pattern this was not so simple because you’d have to look in several places to find the right files. It was easy to forget a file or not think to include a dependency.
* Less browsing for files. Your workflow is simplified because you won’t have to look far for related files. Everything you need for a feature is located in one directory, making switching between them very simple.