QA Decisions Guidance

Review of user stories

Throughout the duration of a sprint, we will check multiple times with the user stories we have created and pushed for finishing in a sprint. This includes: detailed and concise user stories and acceptance criteria created before a sprint and refining that content before the end of a sprint in case:

- We have decided further down a sprint that a user story can be split into several smaller user stories
- Our team has shifted our strategy on a task we have yet to complete and that strategy does not fit with the user stories/acceptance criteria

Functionality testing

After the implementation of a new functionality we will test the data input and output so we know for sure the function does its job. We won't use any external tools for testing this. For example going step by step through the node management system. Testing all the functions and making sure they work according to the user story returning the right data and without major bugs that will heavily affect the application.

Commenting

We comment on methods whose names aren’t self-explanatory. For example a comment that is unnecessary:

```
//Edit admin key
@PutMapping("/editKey/{key}")
public ResponseEntity<Collection<AdminKey>> EditAdminKey(@PathVariable String key, @RequestBody String newValue) {
    return new ResponseEntity<>(adminKeys.EditAdminKey(key, newValue), HttpStatus.OK);
}
```

A comment that is necessary:

```
// creating a new iBeacon and adding it to the iBeacon list
newIBeacon = () => {
    const item = this.state.iBeaconList;
    const id = this.state.iBeaconId;
    const x = 0;
    const y = 0;
    const type = "iBeacon";

    item.push({id, x, y, type});
    this.setState({iBeaconList: item});
    this.setState({iBeaconId: id + 1});
}
```

The basics rules of commenting are simple:

- Make them brief
- Keep them relevant
- Use them liberally, but not to excessive

Unit testing

For unit testing we'll add automated tests, we'll use JUnit as a tool for unit testing. This tool is highly regarded in the software community.

When are we unit testing? For a method that only uses standard Java code for example: List getters, setters and for loops we will not unit test. This is unnecessary since these methods are too standard. A good unit test would be a unit test that tests for example the routing algorithm method that we will create. Since a lot of calculations are in here that we created and we have to assure to work.

Integration testing

When we are in the sprint where we have connected the mobile application to the Spring server, we will apply integration tests.

We test for the following risks:

- Tracking of IBeacons
- Triangulation
- Routing algorithm

Unit testing

What are we unit testing?

We test if the component actually renders, if certain elements in a component render, like headers and text. Besides that we also test functions like login and register.

How much coverage do we have?

Logic coverage

The backend of the Guidance application mostly handles gathering and persisting raw data, at times even processed information. About 20% of the developer defined **functions** are pure logic based and exactly that percentage of functions have been tested. This number may seem low, but the remaining 80% of the functions are handled by imported packages that handle data directly, without taking the developer's time to create such interface from scratch.

Out of these testable 20% of code, about 90% of the **statements** have gone through a successful unit test run. Additionally, **branch** and **condition** coverage is 100% unit tested. Every possible end point of a function has been accounted for. **Lines** of code tested barely scrape the 80 percentile of the unit tests. This is due to slightly refactoring some of the functions so they can be effective, production ready and testable at the same time.

In conclusion, the overall logic coverage would be merely enough. But since the Guidance Team uses already imported functions, the percentage requirement for adequate logic coverage goes down as well.

Front end component coverage

Every component is snapshot tested, all the components that contain logic are thoroughly tested.