Software project documentation

Severi Tikkala

# Background and Current Status of Development

The software project will be related to a new startup company, eloSpaces, where I’m one of the founder. While the most important outcome of the project is to widen my personal skill set, it will also help the company by providing an actual prototype. The company consist of 5 persons both from Finland and China. My responsibilities are focused on the technical concepts and development.

eloSpaces focuses on smart spaces and everything related to smart living. Our first product is eloSeed, an inflatable capsule which creates a personal space around the user. This space will be fully controllable and synchronized with the user. Embedded sensors, actuators and controllers adapt the environment according to the users needs. Measured data from the user and the environment will also be stored in a database for later analyzing and visualization purposes. More information and pictures can be found on our company website <www.elospaces.com>.

The product development is split into separate categories: mechanical structure, embedded electronics, user interface and web & cloud services. The mechanical structure of the product is developed mainly in China and at the moment they are focusing on the manufacturability and how to embed the electronics to the structure. The two most promising solutions to integrate the electronics are “pillowcase” of cloth on the product which contains the electronics and the other is new Nano carbon technologies which are available to us through a Finnish university.

Embedded electronics have been prototyped with prototyping boards such as Microduino (tiny Arduino) and Raspberry Pi (PI). The main idea is that there is one control unit for each capsule which collects the sensor data, controls the devices and communicates to the user and database. At the moment there are few separate parts which will be integrated together and into the capsule. These parts consist of lighting, speakers and basic sensors. Temperature, air pressure, moisture and motion sensors are read by the PI and written to the database or to a local storage if the internet connection is missing. Sensor data resolution is between 100 ms to 10 s depending on the sensor type.

For user interface I have developed a multiplatform mobile application with Apache Cordova (which enables the use web technologies such as html5, css3 and JavaScript instead of native languages). The goal of the user interface was to have one remote controller for all the devices with clear controls and visualization of the important data. However, we have decided that our product is best fit with more flexible solutions than a separate mobile application. ZeroUI and other user interfaces are being looked into to find the best way of interacting. The vision is that the environment would adapt automatically to optimize the environment for the user. This can be achieved through the measurements and analyzing the data. So at the moment the user interface is not highly related to this project.

The cloud services prototyping is at the very early state. There is a mongoDB located in the Microsoft azure virtual server and the schema for the JSON is somewhat decided. Otherwise the development has not yet been started, thus it is a good scope for the school software project and can be done separately from the other prototypes. Only connection to the capsule is the database where to sensor and user data are stored.

# Proposal of the software project

The goal of the software project is to develop a working full stack one page web application for mongoDB sensor data visualization. It will consist of the server backend where data queries, user authentication and rending are done and the client frontend where the data is presented in an informative way. As the software development has heavily shifted from low level coding to efficient usage of open sourced code and frameworks, it is ideal to use as much as possible the frameworks and their examples for the development. However, to maximize my own learning and understanding, it may be better to develop some components from the scratch and not to use existing solutions. The items below show the phases and their content for the project.

* First the specification based on research. All of the items depend on each other.
  + Architecture of application
  + Technologies for development: templates, frameworks
  + Features to be developed
* The development based on the specification
  + Feature driven development to keep software working all the time
  + Unit tests and some ideas from test drive development to keep the quality good
  + Setting up the development environment
* Documentation updated constantly according to the progress
* All the materials for the project are stored in GitHub repository.

The first task is to define the user requirements and write the feature descriptions based on them. The features are used to find the suitable tools and platform to build them on. This means researching the modern web technologies and their fit to this project. Some of the frameworks use very different approaches and that will affect the software architecture dramatically. That’s why both of these items should be kept in mind in the beginning.

After the features and structure are clear, it is best to use some effort to have all the necessary tools for the developing environments. By this I mean tools such as automated build scripts, unit tests, debuggers and other tools. Some of the frameworks offer these tools as the part of the solution, but many of them require 3rd party solutions. This might sound obvious but I have learned that it is a waste of time to develop anything without proper tools.

After everything is set up, the application is developed feature by feature. This should keep every iteration of the application working and it is easier to keep the focus this way. After each step/item is done, this document is updated and in the end returned to the school for grading together with the developed application.

# Specification

## Features

As the purpose of the application is data visualization, a simple dashboard view will be used to show all the relevant data. An example of dashboard view is shown in figure 1.



Figure Example of a dashboard view (<http://www.cyfe.com/images/dashboard-startup.png>)

In the project application the data will consist of user and sensor data, both connected to a unique user. The data is loaded from a database. The key features are listed below.

* User authentication (not secure one)
  + Login with user name and password
  + Creating a new user (Additional)
* Connection to database and Data visualization
  + Loading all the available sensors according to the user logged in
  + Showing one sensor in line graph at the time (See sensor data…)
  + Basic information of user
  + Averages and key figures
* Sensor data visualization
  + Tabs where you can choose the signal to be shown
  + A zoom functionality (additional)
  + Live data feed (additional)
* User settings (additional)
  + Customization options for charts and figures: for example colors
  + Style changes are saved to the database
* Multi-client support

## Architecture and Frameworks

As discussed earlier, the modern web development is heavily based on third party components such as libraries, frameworks, plugins and templates. In this chapter the most important selected components are explained together with the architecture of the application. Later there will be a list and short explanations of the other packages used in the project.

The solution is divided to two different applications: server and client. I made this solution as it felt more flexible to scale up and also it provides clear distinction between the user logic and the backend logic of the server. However, divided applications caused the architecture and communications to be little bit trickier. The communication between these applications is made by using web sockets. JavaScript library Socket.IO was used to implement it as it is well tested and functioning solution.

The server application is built using Node as the platform and Redux as the base of the architecture. Node is very widely used virtual engine for JavaScript especially used in server side hosting. Redux is a new way of web programming. It is based on Facebook’s Flux implementation of stores and one way data binding. I will be going to Redux separately later as the client side architecture is based on the same package.

At the moment server is responsible of the database logic. It handles the connection to the database using the provided username and password for the client, and also fetches the data according to the parameters from the client. The solutions is currently lacking all of the security functions to encrypt data and to check in coming connections and so on. These features will not be implemented as this is a minimum viable demo. Server allows multiple clients to connect and uses the client socket ids to identify the data requests and serve only the correct client.

The client side is built with webpack which is a tool to bundle static components out of the sources files and serve them as a website. The other main component it React, a library for building user interfaces for modern web. It is developed and hosted by Facebook. The main reason I chose React was the fact that it supports server side rendering and uses virtual DOM which highly optimizes the rending of the contents. There were few other options such as Angular and Ember, but after researching forums and blogs it seemed that React is the top notch technology currently in the web development. I had some experience from Angular but the new version Angular 2.0 is not yet released and thus I didn’t want to choose “old” library as the main building block. React is very modular which was good for the dashboard view structure.

Also because Facebook develops Flux and React together, it was logical to use similar architecture in the client application. That is why I chose Redux. The idea behind this architecture are simple states and pure functions. The full documentation of Redux can be found in <http://rackt.org/redux/docs/introduction/index.html>. In short, Redux consist of actions, reducers and a store. Actions are just JS objects with type string and other user defined data. Action creators are functions that simply return an action. Reducers which use the previous state and the action to build a new state which is then set. And the last part is the store which holds the application state. It can tell the current state (getState()), update the state (dispatch(action)) and register listeners via subscribe. Subscribing to a store gives you a notification every time the state is modified.

So Redux is just state machine which is tailored for the web development. The idea is immutable data which means that the old state is never modified, but a new one is created based on the given action and the old state. This allows for example undo/redo functionality with only few lines of code because the states can be stored easily. The server side redux store is actually not used very much, it only dispatches simple objects to the clients who have their own state. When more features will be made on the server side the Redux store will come in handy as it can store server state and part of client states there.

One of the most problematic parts was to include asynchronous actions between the redux stores. Every action should have three states: pending, success and failure, these states should be automatically updated after promises are resolved. For this there were a few libraries that include appropriate Redux middleware. Middleware is a layer where actions can be modified before they are dispatched to the reducer. For example redux-thunk layer would have the required functionality but as the solution is divided to two applications I couldn’t make it work. Thus, I made the asynchronous with my own states and promises. It is not as flexible and easy to maintain but it works for this project.

Figure 2 illustrates the software architectures of the both applications. Yellow blocks were given a thought during planning but are not implemented during this project.



Figure 2 Software architecture of the server and client applications



Figure 3 Simplification of the program states and data/action flow

Figure 3 shows the simplified version of the client’s state machine. The client application starts with the empty screen, only allowing the user to enter the login data. After entering the data the server tries to authenticate the user and connect him to database (1.). In this phase there are multiple possible errors from wrong user data to connecting with the server where the database is located. If some if these errors occurs, error will be returned to the client which will display it and the state is set back to not connected (0.).

After the connection is successfully done (2.), the basic user data will be returned to the client and loading action is dispatched to automatically load the data of the first sensor. Every time a loading action is dispatched, data for the selected sensor is fetched from the database. The application will not hold other sensors in its local memory. If the data is successfully loaded from the database, it is sent to the client and *loaded* state (4.) is set. If loading fails for some reason, the *loaded* state is still set, but the data returned is an error containing information instead of the actual data. The *loaded* state is only used to show that the server has answered to our loading request, not to indicate that the load was a success.

From this state (4.) the user can disconnected from the server and the application will return to the (0.) state or the user can load different sensors by dispatching load requests. Refreshing the browser site will set the application to its starting stage (0.).

# Additional Packages

There were some interesting new packages related to web development and especially to the new versions of the JavaScript language.

* Babel
  + A package that allows the usage of ES6 and ES7 JavaScript standards that are being developed but not yet supported by many browsers. The package translates the new syntaxes to the older ES5 syntax. Examples of new features are array function syntax () => {}, classes and many others.
* Foundation-Sites
  + Styling sheets with a mobile first ideology.
  + Used in Sass to have more flexibility
* Immutable
  + Immutable data collections. Extends JavaScript objects and adds the immutable features to them. Meaning that once immutable data has been created it can’t be modified. This goes very well with the idea behind Redux.
* Monk
  + An additional API layer for mongoDB to improve database usability.
* Promise
  + Promises for asynchronous operations that replace the old way of callback functions passed. Allows you to do AsynchCall().then() and AsynchCall().onSuccess/onError very simple without long callback chains.
* Chai and mocha (and jsdom)
  + Mocha is a unit test framework
  + Chai is a TDD assertion library
  + Jsdom is used to emulate the DOM during unit testing to the also the UI components
  + Together they build up suitable environment for testing the application
* React-ChartJS
  + React layer for ChartJS charting library which is used in the project. ChartJS had so missing functionality and bugs that affect the project at the moment, but new major release ChartJS 2.0 is soon released and should fix most of the problems.
  + There are many many other charting libraries that I might choose to use when I have more time to do research on them.

# Usage (hosted)

The application is not currently hosted everywhere.

# Usage (locally)

As there are two separate applications there are two different options to start the program.

Either separately by running

“npm run start” – for the server (schoolcourse/serverside/)

“npm run devrun” – for the client (schoocourse/clientside/)

Or with one command

“npm run both” – for both concurrently (schoolcourse/)

After the client is running you can connect to it with a web browser by using “localhost:8080” URL. The sockets will use the port 8090 for the communication. The user interface on the client will indicate if the connection with the server is active.

The server side will log it actions and states in to the command line console. The client side will be logging additional data to the web browsers console which is available somewhere from the menu and “developer tools”, depending on the used browser.

# Learning Outcomes and Comments

First of all I have to admit that the modern web development is a mess. The amount of possibilities when choosing the platforms and frameworks is endless. Very few clear standards exists, thus my background from programming C according to strict standards feels weird. However, the big corporations are offering their own solutions from which many have been open sourced. Good examples of these are Node.js which build on Chrome’s V8 JavaScript engine and React.js which is developed and used by Facebook and Instagram. So in the end there are only the few platforms left that one should choose.

Before and during the development I watched hours of YouTube clips and read many blogs, tutorials and other guides related to the topic. At first I felt that especially React and Redux are very complex and inflexible, but as I tried myself quickly started the get on the basic principles. A big help was this tutorial written by a fellow Finnish man: <http://teropa.info/blog/2015/09/10/full-stack-redux-tutorial.html>.

I started the development by following his tutorial and after understanding the concepts I started to go into my own way and own solutions. There were few moments where I lost all hope as the code just didn’t work or I didn’t understand anything from a blog post, but in the end I can say gladly that I have built a full stack solution. And most of all I have much wider understanding of web development and its complexity.