15.12.2015

Lappeenranta University of Technology

School of Business and Management

Department of Software Engineering & Information Management

Otto Laitinen <otto.m.laitinen@student.lut.fi>

Lassi Lääti <lassi.laati@student.lut.fi>

Anna Osipova <anna.osipova@student.lut.fi>

Project Report

CT30A9301 Code Camp on Platform Based Application Development (4 cr)

Creation of the Paavo Sähkö

Source Code:

<https://github.com/lasshi/otso-cc>

1 Case

In this code camp we were supposed to solve a case by building an application based on the case material. Case was provided LTC-OTSO, a company in Lappeenranta. The client was Lähitapiola, an insurance company who is looking into expanding it's services and building a network of companies.

The problem for the insurance company is that when various tasks are outsourced the blame for bad customer service still falls on the company itself. Despite all the work done by the insurance company behind the scenes the end customer only sees the tip of the iceberg: the service provided by the outsourcing company. To improve customer experience Lähitapiola wanted to implement a service to help evaluate and manage subcontractors.

The case had three main aspects and each team was allowed to choose to implement some or all of them.

- A mobile application or a web page for the customer to give feedback on his experience that would help solve the problem of encouraging customers to give feedback
- A mobile or a web application to help manufacturer manage subcontractors, view the user reviews, create orders
- A mobile or a web application for subcontractor to browse his performance and see the scoreboard

The only technical requirements were that the application was to be done using web development tools and it was to be ported to a phone using PhoneGap. It was recommended to use jQuery Mobile UI for the frontend.

2 Designing the app

After hearing out the case details our team had a brainstorming session. As a result we decided to build one app that would fulfill all three main case requirements. We decided to put an emphasis on user interface and user experience, because we believed that these aspects are crucial to good customer experience and it will help encourage customers to give feedback. Our three main goals concerning the case were going to be to:

- Create a mobile application or web page for the customer to give feedback on his experience
- Create a mobile or a web application where the manufacturer can manage his subcontractor
- Create a mobile or a web application where the subcontractor can browse feedback for his work or services and see a scorecard of his performance

When brainstorming the task further we decided to split the functionality into two applications using the same server backend.

- web/mobile application where
  - manufacturer can login and manage all subcontractors and orders
  - subcontractor can login and see his/hers orders and scoreboard
  - different content is displayed based based on logged in user
- web form for customer feedback
  - unique url for customers to fill in feedback form

Server backend would take care of data communication for both of these apps. It would do the following:

- email unique feedback urls that are linked to orders to customers
- store feedback data
- store subcontractor and order data
- store application user data

2.1 Chosen technologies

We chose NodeJS and Express for the backend functionality for several reasons. Since we are working on a mobile environment, it is important to note that connections can't always be lightning fast. Conventional apache based PHP + MySql backends use synchronous communication, which means that the database waits for the query to be complete, before processing another one. When there are lots of users, with small bandwidth, the backend can get slow. Instead when we use Node JS, the asynchronous nature of callbacks in callbacks makes simultaneous queries possible, which means that the backend doesn't get overloaded that easily.

MongoDB is the database of choice for the backend. However due to time limitations and having to prioritize the most important aspects of the application we did not implement the database functionality in this version of the prototype. Instead we decided to use the simplest method of data storage for now and store the data in JSON in text files on the server.

ReactJS was chosen for front end of the manufacturer/subcontractor app to provide seamless transition between pages. Javascript allows adding and removing of DOM elements from the page without refreshing it. While jQuery would have allowed us to do that, it is easier to use a Javascript framework with routing and templating to not have to add and remove elements manually.

Bootstrap and Google's Material Design were chosen for the interface, because they allow to create scaling applications that look good on different screen sized. This was important in our case, because our app was supposed to work on both web and mobile browsers. Another reason to choose these UI frameworks was because they provide good looking interfaces with little designing effort needed.

We used Git for source control. This allowed all of us to work on the code at the same time and later easily merge all the work.

Webpack was used as a building tool to merge all the Javascript modules and libraries together and minify them. Javascript code minification is used to reduce files sizes and speed up page loading.

3 Implementation

Implementation started with listing all the functionality that our applications were going to have. This basically allowed us to break the applications into components and design each one separately. These components were:

- Login screen
- Manufacturer dashboard (starting page)
- Subcontractor list
- Individual subcontractor page
- Order list
- Subcontractor's order list
- Feedback form

Next important step in designing the architecture was to design the data structures. With the list of components it was clear what "tables" and fields we would need. We proceeded to implement the server's data accessing interface and the REST API to give away data to the web application. We started with implementing the basic GET functionality for the data. Then the POST functionality for submitting feedback forms and creating orders was created. Later added a few more queries that proved to be useful to have.

At the same time as the basic backend functionality was being created by some of our team members, others started working on the web application. For each of the components we drew a mockup of the screen view with all the elements. We outlined how these view will be connected together at what UI elements will be needed for that.

We implemented each view as a separate component in React. Some of the views required several child components for rendering and updates to work smoothly. All the data is fetched from the backend via Ajax calls.

After implementing the basic functionality we tested tested the application to see, first of all, that everything works, but also to identify usability flaws, and corrected the functionality based on these flaws. Every application use case should be straightforward, easy, clear without reading manuals and require the lowest possible amount of clicks.

Last step was to make the application look good, and most importantly, to scale on different devices. At this point we ported the application to our phones with PhoneGap and repeated the usability testing. This showed some new mobile specific phones, most importantly that wide tables did not fit on mobile screen and that the menu bar looked better on top of the screen on wide browsers but on the bottom on mobiles. We used Material Design to take care of table scaling and added some viewport-specific CSS to take care of the menu positioning.

4 Complete Application functionality

In the end we were able to implement all of our planned features, with some extra features. We have functionality for basic customers, subcontractors and manufacturers. The basic logic of the application is that a manufacturer can make an order and assign it to chosen subcontractors. After the work is done, a custom feedback link is sent to the customer as an email, and the given feedback is connected to the relative subcontractor. After this loop has ran a few times, the manufacturer can review their subcontractors based on given feedbacks. Manufacturers can also sort their subcontractors in different orders, to see e.g who has the best ratings.

5 Future development

For future development, we listed following things to be things to be polished

- Appearance
- Rewards and gamifying customers experience
- Gamifying customer experience
- Graphic data representation
- Optimization for large amounts of data
- Safety

Our team had only five days time to create our application for customer presentation, so we cut some corners when creating user interface. Our whole application is constructed on grey and white, and there is only pre made open source icons provided by Google. Application itself doesn’t have own logo, so it kind of doesn’t have own identity. Of course, after adding real subcontractors, subcontractors would have proper sites with their own logo, so they would stand out a bit. Pages don’t have good animating when transitioning from page to another. This is possible with React animation classes, and we researched it but didn’t have time to implement.

Right now subcontractor experience is gamified. Our vision was to also implement this system to also customer space. Idea came from internet shops, for example cdon.com. In cdon, customers are given rewards on when they do spend money and when they give feedback on products that they have bought. For example verkkokauppa.com also does giveaways between customers who review products that they have bought. On our database we can merge orders and feedbacks based on email addresses that customers have provided us with. We have already ready queries on our server for fetching information on customers. But because it wasn’t part of the case, we didn’t implement space for customers.

Our database is right now built on top of Node.js server, reading data from JSON files. This is fast to implement and is possible to grow into a more mature system. But to build more reliable and safe system, proper database could be advised. Right now doing queries doesn't expect user to authenticate themselves, so basically anyone could read whole account.json file in plain text just with HTTP GET. If this project would be developed further, we would go and set up no-sql mongodb and probably setup passport authentication for our users. This way we could keep user information safe(r) from attacks. Although there is no such thing as completely safe system, having data like authentication passes as plain text is never advised in production software.