Technical University of Denmark

Bachelor project in Software Technology

Cloud Based Social Network for Local Communities

Student
Patrick Løgstrup Olesen – s103256

Supervisor
Christian Damsgaard Jensen

August 2014

Technical University of Denmark
DTU Compute
Department of Applied Mathematics and Computer Science
Matematiktorvet, Building 303 B, DK-2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

The purpose of this report is to describe and present a social network, which is specially targeted at local communities. The reason behind targeting local communities is because having a good relationship with ones neighbors is a good way to get social interaction, a sense of security and to share resources. All of which are things that are practiced in a decreasing extent, since the uprising of digital media.

In order to make the framework available for as many people as possible, it were developed in HTML, JavaScript and PHP. Due to the increased use of both tablets and mobiles, which are sometimes on rather slow mobile connections, it was designed as a single-page application with the use of the libraries KnockoutJS and SammyJS.

Even though the product was not finished at the point of hand-in, there are still a lot of positives, especially the server, which is almost fully functional and ready for servicing a lot of local communities. Even though this social network is not the direct solution to the problems, it can still make an impact in training people towards getting some good habits, which can ultimately lead to better life quality.

This page intentionally left blank

Acknowledgements

I would like to thank a few people for the help and support throughout this project.

First of all a special thanks to Christian Damsgaard Jensen, for supervising the project, and making the Solar Decahtlon experience possible for the Compute students at DTU. I could imagine it being hard enough to supervise 1-2 students, then multiply that with 3.

In addition, a special thanks to the company 7it, for being patient with me during the last couple of weeks of the project, and for making their servers available for the live project. In addition to that, a special thanks to the head of 7it - Ruben Jusiong, for mental support and invaluable advice.

Also a thanks to all members of Team DTU, for the unique experience in Versailles. An 8th place out of 20 is definitely something to be proud of.

A special thanks also goes out to Carsten Nilsson, for corporation and good team spirit throughout the course of the last 5 months.

This page intentionally left blank

Preface

Kongens Lyngby, 2014-08-22

Signature (Patrick Løgstrup Olesen)

Table of Contents

Chapter 1

Introduction

During the last few decades, more and more people are moving from the suburbs into the capitals and major cities. This puts a strain on the transportation systems and increases the densification in the areas. But even though the areas is getting more dense, studies show, that people in general are feeling more lonely than ever before. Even though researchers are having a hard time settling whether or not social networks are to blame, it is widely accepted that technology plays a key role in this development.[1]

The purpose of this report is to describe[2] and present[3] a software system, which should be able to engage the residents in small neighborhoods to share resources, socialize and get to know their neighbors better, which have many advantages. The platform is built around a case study, for the international competition Solar Decathlon Europe©, which was made by the Danish team from DTU (Technical University of Denmark). The case study concerns a special type of plus-energy-houses, which Team DTU designed and built, called EMBRACE. Even though the system is developed for EMBRACE, it is not intended to be limited to only these type of residential dwellings. The platform should be available to every type of household, but of course, extra features will be added for EMBRACE houses.

The platform supporting all the EMBRACE features is a cloud solution, which was developed in collaboration with 6 master's students (1 working on special course, the others doing their thesis). Due to the cloud solution only supporting EMBRACE houses, it was necessary to make a standalone solution, which would support all other households. That solution is what the report will primarily focus on, as well as the integration with the cloud.

Some of the challenges in building a social network system, is the fact, that the market are currently getting dominated by three large companies; Facebook, Twitter and Google. Therefore it is crucial for this project to form its own identity. Another crucial part of the system is to make it intuitive, responsive and easy to use. This is very important, since people today do not have the attention span for a steep learning curve, or for that matter the patience to wait on the internet connection to come back, during a page reload. Due to the subjective nature of the mentioned measures, the only objective measurements are of the backend system, which will be measured in terms of speed and memory.[4]

[1] These claims will be established in Chapter 2 (Background)
[2] See Chapters 3 and 4 (Design and Implementation)
[3] See Chapter 5 (Results)
[4] See Chapters 5 and 6 (Results and Evaluation)

Chapter 2

Background

The purpose of this chapter is to elaborate on the claims made in Chapter 1, with surveys, statistics and scientific reports. [WRITE MORE HERE WHEN YOU ACTUALLY HAVE WRITTEN THE SECTION, DUDE!] Okay let's just write a lot of stuff in order to push some of the other stuff a few lines down. I cannot imagine it possibly going longer than this. I think this is fine. Just need some more words to look good.

> "I believe in innovation and that the way you get innovation is you fund research and you learn the basic facts."
>
> **- Bill Gates, co-founder of Microsoft**

Rise of the Machines

Since the surge of what is known as the World Wide Web back in the mid-90s, software technology has been making a larger impact on the world around us. Ever since 2007 there has been a trend with more people studying computer science than the previous year [1]. This is also observable in Denmark, where the grades for being accepted at the software civil engineering line at DTU (Technical University of Denmark) have increased by more than 50 % during the period 2010-14[5] [2]. But this trend is not only on an academic level. From the period 2000-2013 the world has gone from 360 million users of the internet, to almost 3 billion users [3], which is equivalent to almost 40 % of the world population. In Denmark 92 % of all people between 16-74 years are online [4] and as Figure 1 shows, there have been a huge rise in the usage of social media since 2005 [5]. All these figures illustrates the size of the market, as well as the development going forward. People are getting more and more used to having machines around them, and logging their behavior and actions on these devices. This is one of the reasons data mining, big data and business intelligence are so popular, because behavioral data is very profitable. But not only has these devices changed the way we think, they have also changed the way we socialize and interact with others. But is that a good or bad change?

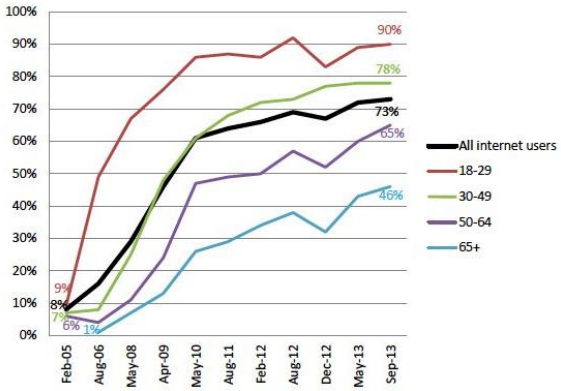


Figure 1: The percentage of internet users, who use social networking sites, since the release of Facebook.

> "If you're not paying for it, you're not the customer; you're the product being sold."
>
> **- Andrew Lewis, known for this quote online**

[5] Going from 6.1 to 9.6 on the Danish grade scale, which ranges from -3 to 12.

Psychological Impact

Studies show that more people feel sad than before [6]. More people communicate through text, which means less body language, which is equal to a poor social experience. This can be one of the reasons to explain the trend, the lowered communication quality. Something else, which might be to blame is the increased use of technology. It has been shown, that people using social media, are 30 percent less likely to know their neighbors, and 26 percent less likely to provide them companionship [7]. Besides the same study does show signs, that social media increases diversity in peoples friend base, which is a positive sign.

Less is more

One of the reasons social media can be to blame, is due to the nature of the communication on the social media. You have a lot of weak ties [6], which basically mean you know a lot of people superficially, but this will strike hard, in times of struggle, where the good friends are necessary in order to help out. So what does this mean? Well, it means that sometimes "less is more". Less friends, but stronger ties, can bring benefits along the way, and that is simply not the concept of Facebook, which rewards people for having more friends, through their several games etc. But what does this have to do with neighbors? Well neighbors are a quick and easy way of getting that social interaction, when needed. Friends might be several hours away, but the neighbors are right there. Maybe you just want to watch something in the television with someone, then the neighbor is just there.

Community Counselling

Besides a lot of positive effects of good neighborhoods have been shown, where people are more likely to trust each other and feel safer in their homes, due to the environment [8].

> When one neighbor helps another, we strengthen our communities.
>
> **- Jennifer Pahlka, CEO of Code to America**

But then it is a bit concerning, that the average person in today's world, wouldn't even be able to select their neighbor's from a police lineup [9]. This really shows how much the attention has been focused from our communities, to the different screens located in our homes.

More "neighbors",

Maybe it isn't just the technology which is to blame. Maybe it is a matter of inflation. The living spaces are getting denser, and the open areas are getting more closed. This is especially true for Copenhagen, the Danish capital city, where it has been predicted in a report from the city, that there will be an expected growth of 33 % in the period 2005-2025 [10]. Especially a lot of young people are coming to the city. But the fact that we know our neighbors less, are still a reality.

Less waste

Another of the benefits attached to getting a closer bond with our neighbors, is the fact, that it is possible to share food and resources with them. Imagine the scenario; "You are

really behind on some work, and need to stay up all night. You come to realize that you need coffee, but once you go to the kitchen, there is no more coffee. The last store closed 30 minutes ago…", the reason this is a good illustration, is because it is something most people have experienced in their lives. Neighbors ringing your doorbell asking for a cellphone charger, a bicycle pump, or maybe even sugar, are just some of the examples of shared resources. Of course it is logical that we should save on the resources, but any way the claims gets further backed up by the fact, that the average human throws out roughly 105 kilograms of food each year [11]. Even in Denmark, there is a new movement, called "Stop wasting food" (directly translated), which shows some of the figures from Denmark. One of the studies there, shows that the average Danish consumer living alone, wastes roughly 99 kilograms of food per year, but as soon as there are two in the household, this drops to a staggering 65 kilograms [12]. Even though it only concerns the household, it still shows, that with the right possibilities and mindset, it is possible to work together with ones neighbors in order to fight the food waste.

This is also why these issues were some of the major subjects in this year's edition of the Solar Decathlon Europe.

Solar Decathlon Europe

The Solar Decathlon Europe, is a competition held every 2nd year. 20 teams, which can consist of one or more universities, from all over the world, design and create plus-energy houses, which are then displayed for the public in the hosting country for a course of two weeks. While being up for display, the houses are also participating in the competition, which measures data from the houses, and depending on the results, points are awarded accordingly. The reason it is a decathlon, is because there are 10 major competitions, which are being awarded. One of the teams which qualified this year (2014) was the Danish team, Team DTU.

EMBRACE

Team DTU's contribution is named EMBRACE[6], and some of the issues, which the project tries to solve, is global densification in major cities, social isolation as well as waste of resources (e.g. food, water, electricity). The idea is to build several EMBRACE houses on top of already existing buildings. This would require the buildings to have flat rooftops, but since most of the angle rooftop buildings in Copenhagen do currently not live up to the Danish regulations, which are to be upheld before the year 2025, there will be a lot of rooftop renovations. The idea was then to flatten the roofs during renovation, making them able to host the houses. For the rest of the report a group of EMBRACE houses on top of a building, will be referred to as *a community*.

Figure 2 – The Danish contribution to the competition Solar Decathlon Europe, EMBRACE, by Team DTU.

[6] More can be read online, at http://solardecathlon.dk

There are several benefits of building these small communities on top of already existing buildings (1) the extra energy, which the community produces will be awarded to the host building, prior to being put on the smart grid, (2) extra room for new residents moving to the city, (3) gives a better urban flow to industrial areas, which would otherwise be "dead" from 17:00 to 05:00.

Word Definition

Community

A small group of EMBRACE houses, which are built on top of already existing buildings.

The Compute contribution

One way of supporting these intelligent houses is to have a computer in the house, which can connect to the IHC systems (Intelligent Home Control), but this year, the chosen solution was to build a platform, which was able to support several households, and add the sense of remote controlling to the home control system. This platform is what is known as the EPIC cloud (EPIC = Energy Preservation and Information Computation). The cloud supports all the houses in the community. The reason this is desirable is because this cuts down a lot on the energy. Instead of having 10-12 computers running to maintain the house control, there is only one running on the cloud system, which is dedicated to that purpose.

The cloud is able to support a lot of different areas, like home control, data harvesting, statistics etc. and one of the branches to this was the social networking system. The idea is to encourage the residents to spend more time with their neighbors, and thus only risk benefitting from some of the advantages, which has been mentioned.

But seeing as the scope of the cloud is only for the EMBRACE houses, it was decided to expand the target group, and also aim for everyone else with an internet connection. In order to do that, it was necessary to create a standalone system, which would be able to integrate with the cloud.

Chapter 3

Design

The purpose of this section is to layout the architecture, and define how the system should work on a high abstraction level. This part of the report is really the foundation of the chapters to come.

"Creativity is intelligence having fun."

- Albert Einstein

System Architecture

Starting from the top and going down always seems like the greatest idea. That way the frames are set for the next part of the system.

The Cloud Platform

The whole idea behind the EPIC cloud, is that it is supposed to harvest and store data from the house, while running the other services as well.

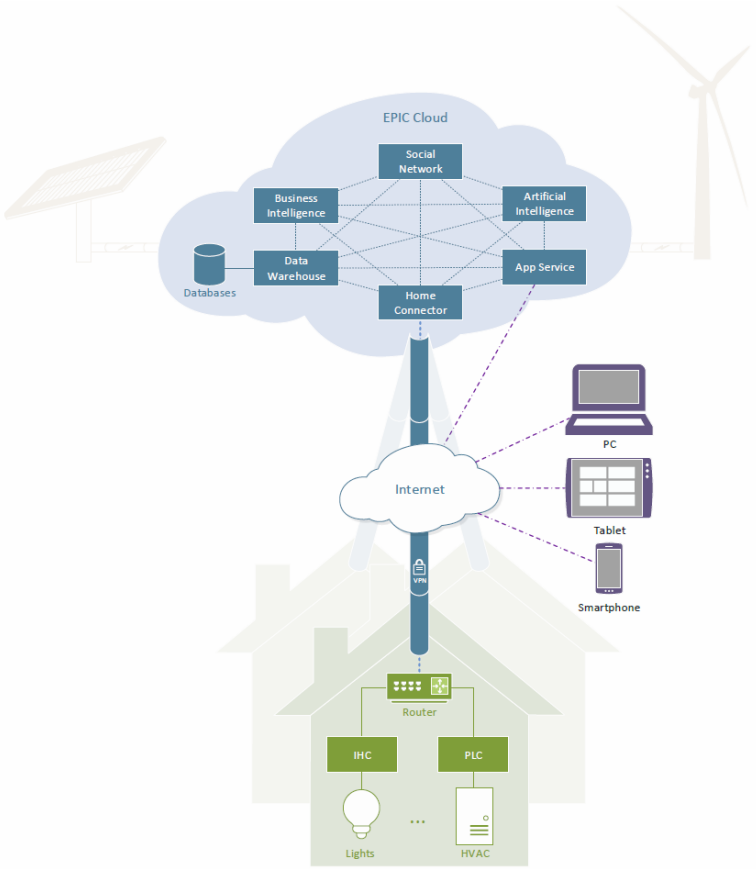


Figure 3 – The conceptual drawing of how the EPIC cloud works. The whole idea, is that you can connect to the cloud via some end points. When running it to access the IHC from the internet, it is the App Service, which is used, but when aiming for the community data, the end points is available through the Social Networking Service. Image Source: Team DTU – Solar Decathlon Europe 2014

The end points, which the website application connects to, are not part of the App Service, but instead part of the Social Networking Service, as illustrated on the following graphic:

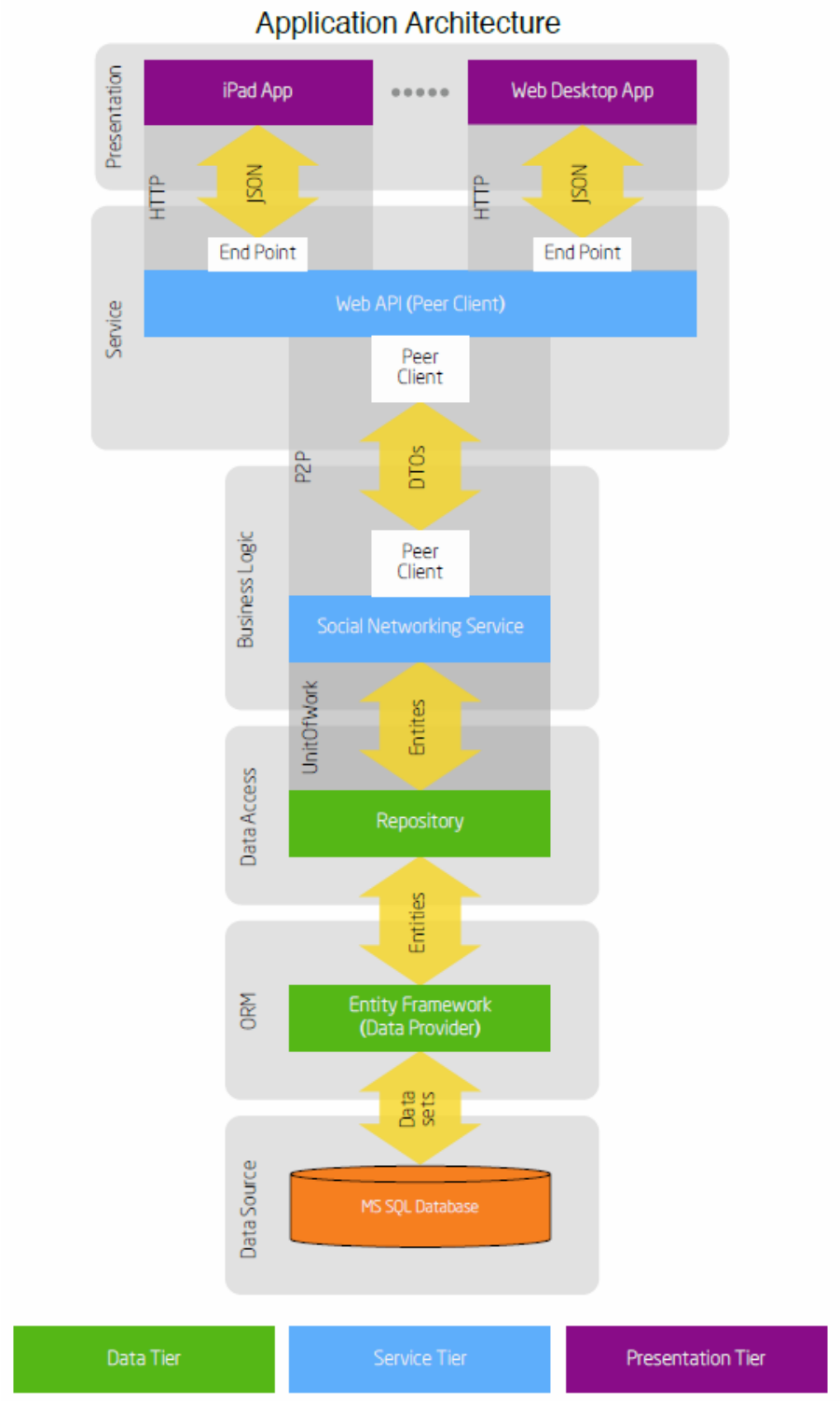


Figure 4 – A visualization of the data flow through the EPIC cloud. For the website application, which are not supposed to connect with the house, the Social Networking Service is the only necessary application layer.
Image Source: Carsten Nilsson

Server / Client Communication

Even though the cloud is of great interest to this project in terms of the integration and synchronization, the most important part are how the modules on the website connect.

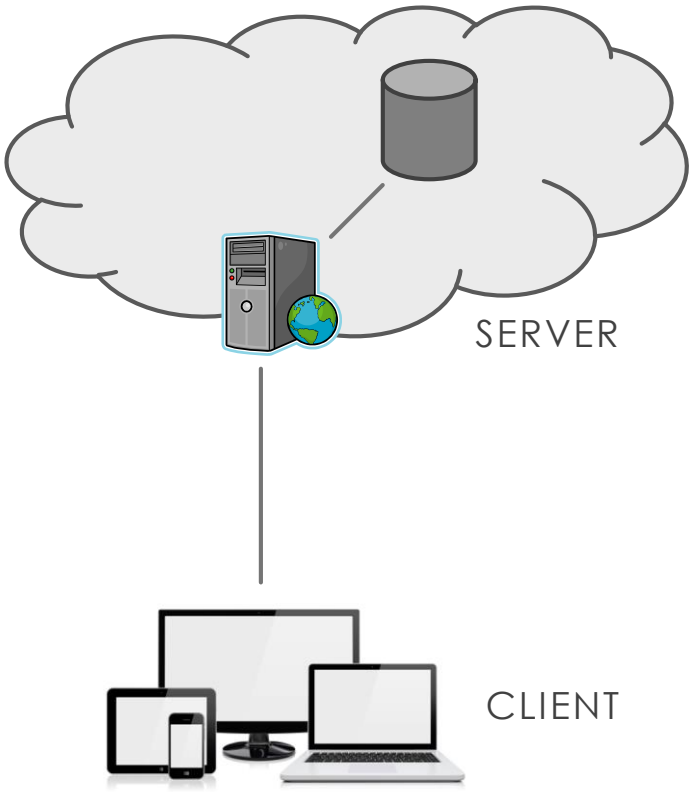


Figure 5 – An illustration of how the server is for computing and connecting to the database. The languages are undetermined.

Overall

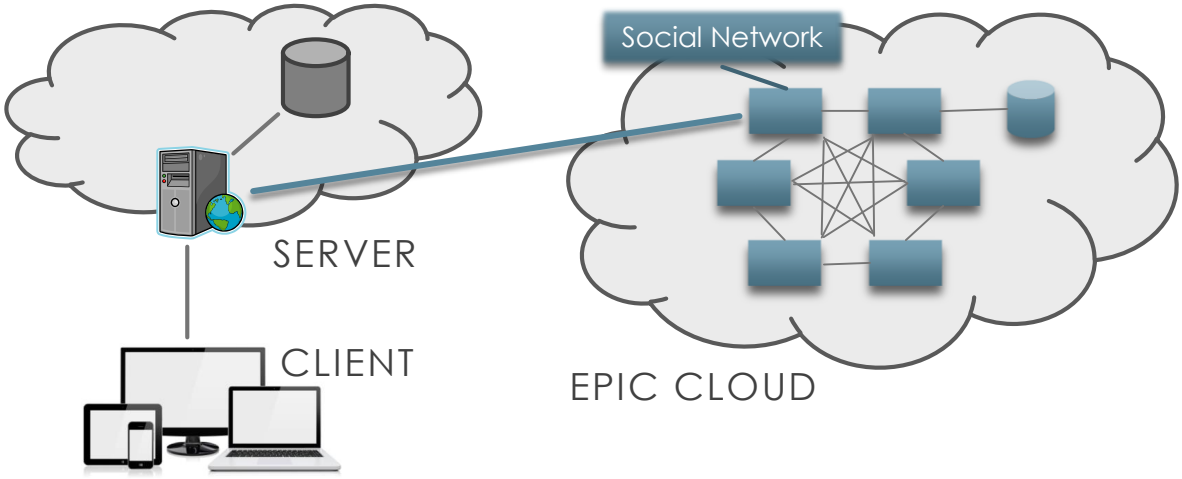


Figure 6 – Illustration of how it is the server, and not the client, which is responsible for connecting with the Social Networking Service.

Single-Page Application

In order to make the experience on the website more fluid, it was decided to go for the single-page application design. Basically a single-page application is a site, which receives all the necessary information in a single page load, and does not depend on a page refresh. The reason this is so attractive, is because most devices today have enough power to run many commands, so the clients are more than capable of being responsible of the page handling, and at the same time, it leaves up for a more fluid experience, due to the fact, that the most basic things, are handled locally, with no interference with a server.

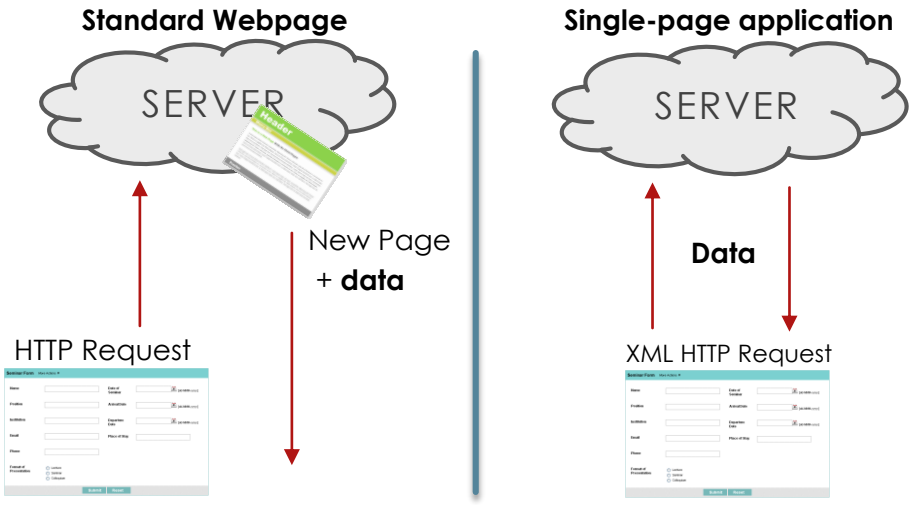


Figure 7 – An illustration of how the standard protocol works when communicating with a server. Single-page applications uses the XMLHttpRequest in order to send data without refresh.

Today, where mobile networks are getting more popular per day, it can be very expensive to depend on page reloads. Let's say the page (plus all the scripts which has to be reloaded as well!) is 1 megabyte, and the internet is temporarily bad. On a single-page application you could risk not noticing at all, because you could be doing other things simultaneously, or at least the overhead wouldn't be as tremendous, and the download shorter, since all the basic things are already in the memory.

Model View Viewmodel

Also referred to as MVVM, is an architectural pattern, which helps controlling the flow of a website according to the concept of having a 'view', a 'model' and a 'viewmodel'. The model is the backend data and logic, which is fed to the client. The viewmodel is kind of the link, between the model and the view, so basically the viewmodel is what interprets the model, and binds the data into the view (the view being what is visible to the user).

So each time the viewmodel gets new information from the model, it automatically gets binded into the view, and the interface updates accordingly.

Some of the known MVVM presentation models are Silverlight (Windows .NET), AngularJS (developed by Google) and KnockoutJS.

Routing

When operating single-page applications, routing is an invaluable feature. Basically it enables the manipulation of the URL, in order for the user to be able to use the 'back' button, while at the same time adding triggers to different URLs in order for the correct behavior to be triggered when entering a specific URL. Some of the known routers are HistoryJS, SammyJS (which was once used on Twitter) and the History API for HTML5.

Model

The model is pretty much an expression for the backend system. Since that is quite extensive, the first thing that has to happen, is a brainstorm, just to get started:

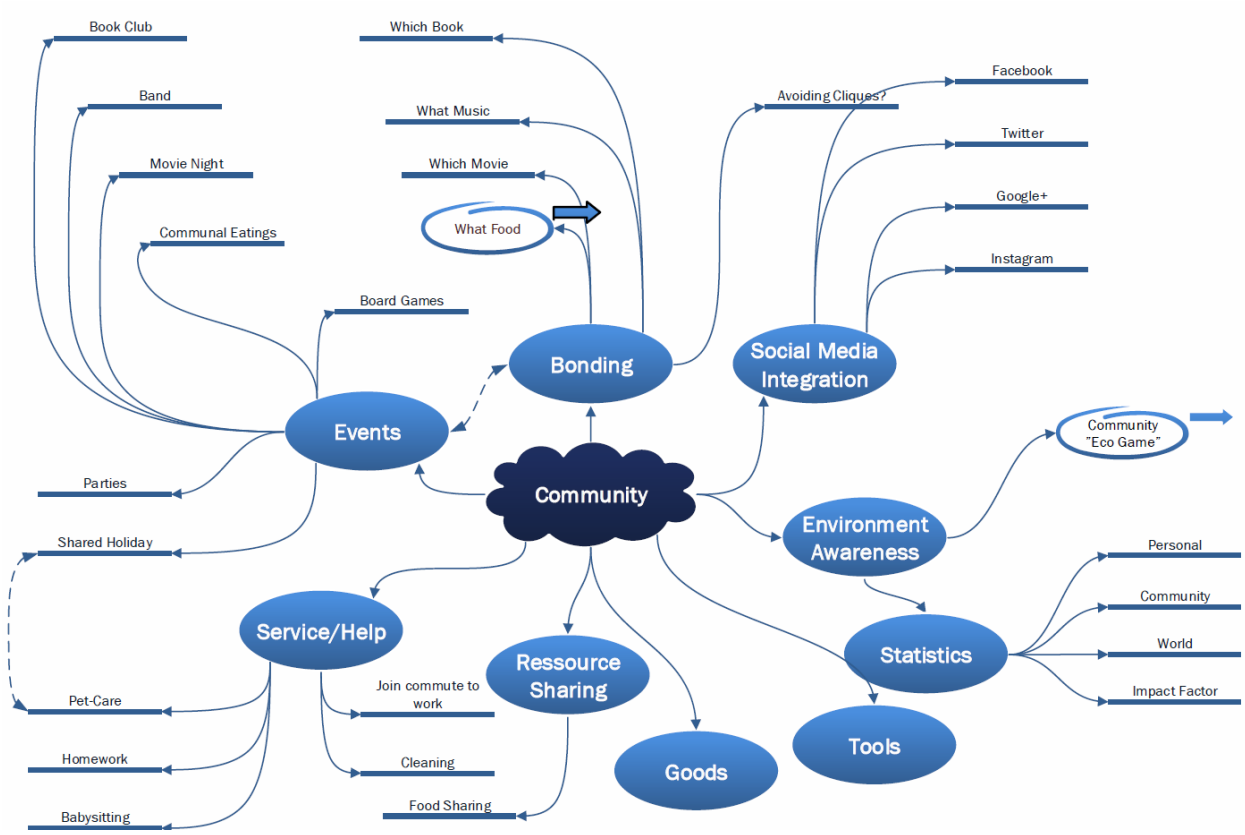


Figure 8 – The early brainstorm regarding the backend system. This was originally intended to be exclusively for the EMBRACE houses, but that changed upon expanding the scope to everyone.

Some of the points passed the test, while others didn't. For example adding your music, movies, books etc. in an online library was quite extensive and did not pass. Another example of something that did not pass is the "Eco Game".

Gamification

First of all, what is gamification? According to Wikipedia, gamification is "the use of game thinking and game mechanics in non-game contexts to engage users in solving problems". So the idea behind mentioning gamification in this context is for motivating the users of the system to socialize more, save more energy and waste less food.

During several brainstorms, the following ideas were selected for the game:

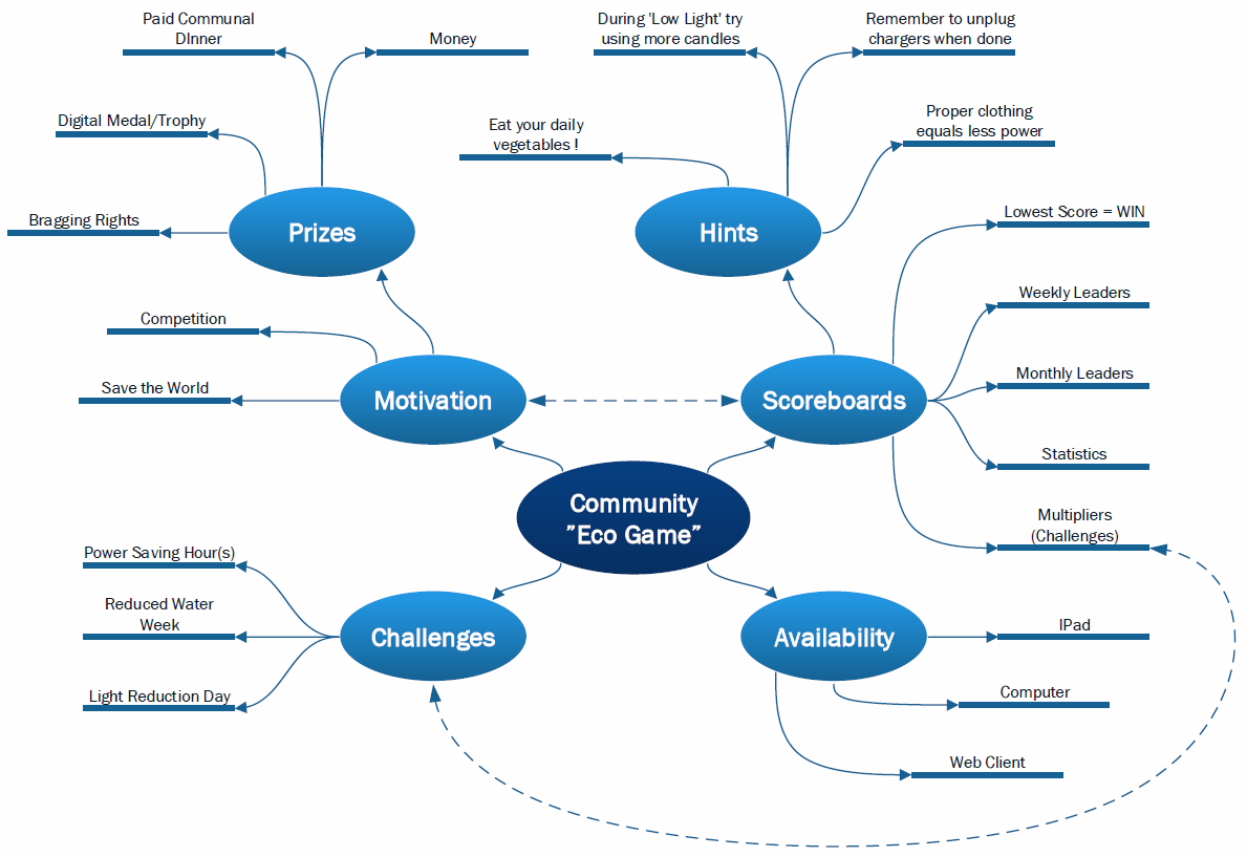


Figure 9 – A possible rendition of which elements the "Game" could contain. Challenges being something which triggers extra points. The idea is to score as many points as possible, and try to motivate the users somehow to keep wanting that

It all sounds good in practice, but during a workshop, it was made clear, that with games comes competition, and with competition comes disagreement, and even though some might argue that competition is healthy, it was decided, that trying to promote a social community, where everyone should be friends, was in conflict with the entire game design.

That was one reason. There were also another one… Cheating! One way of trying to encourage people to socialize was to give them points, depending on how many they visited (using measures from EMBRACE in order to verify the data), and divide the resources used for the statistics. For example User 1 and User 3 visits User 9 for the evening, User 1 and 3 have very little house consumption in this timespan, so actually in the context of gamification, player 9 would be punished for having guests, except if the system divides the total usage for all 3 houses, which overall should be lower than if they would have been alone. Great. Problem solved! No… This means, that User 4 accepts the event, well knowing he is not going to show up, in order to maximize his points.

With that said, gamification is not completely out the window, it was just never decided upon, how to make a game, which could simultaneously make a good environment for social interaction, while also maintaining a competitive spirit towards the game.

Domain Model

In order to get an overview of the project, a domain model was made. The domain model shows the primary classes/entities as well as the relations between them.

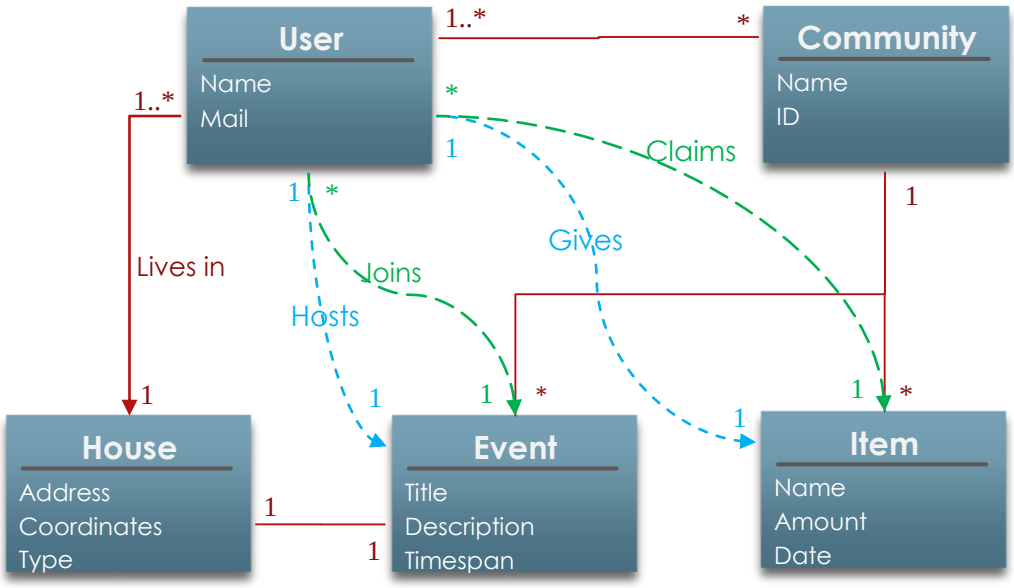


Figure 10 – A domain model, which shows the relations between entities in the domain. For example, an event can only have one host (the person inviting), but 0 to many joining, which is marked by the *. Same applies to the item. Only one can offer/give an item, but many people can claim that item (or at least parts of it, depending on the amount).

Responsive Design

One of the criteria in the introduction is that the user interface is responsive, which basically means it will look good across different devices. But why even bother? Studies show, that every 17th person in the world today owns a tablet, while every 5th owns a smartphone. Smartphones have even surpassed desktop computers today in penetration factor [13]. This makes it VERY interesting, to go for a responsive design.

Some of the measures to achieve this is by having the right strategy off the gate.

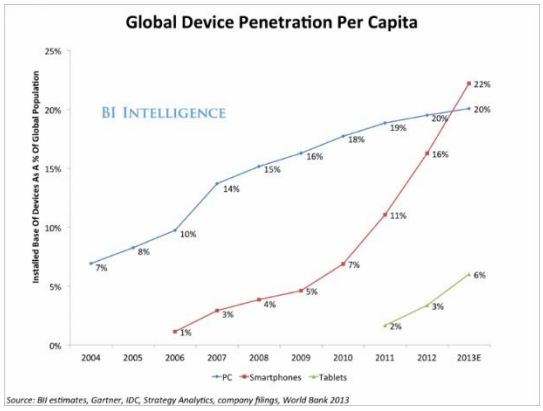


Figure 11 – Showing the impact factor of tablets and cellphones on the market today.

There are two trending strategies at the moment:

1) **Mobile-first approach.**
   This is a bottom-up approach, where the first design is made to small resolution screens, where the most important content is prioritized first, and then as the screen size grows, some of the less important content comes in, or more space is used up. Usually this is implemented using media queries, where some stylesheet rules are left out until a certain width has been reached.

2) **Relative measures.**
   Basically the idea here, is to use percentages instead of pixels, and have everything adjust relative to the size of the screen. If designed properly, the website should look identical on all kinds of screens with the same screen ratio.

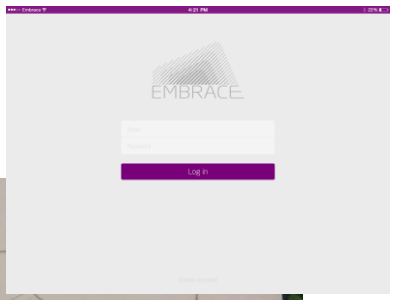
The amount of possibilities are very huge, and confusing. It is easy to get lost. There is a valid third option, and that is to use a system library like Bootstrap.

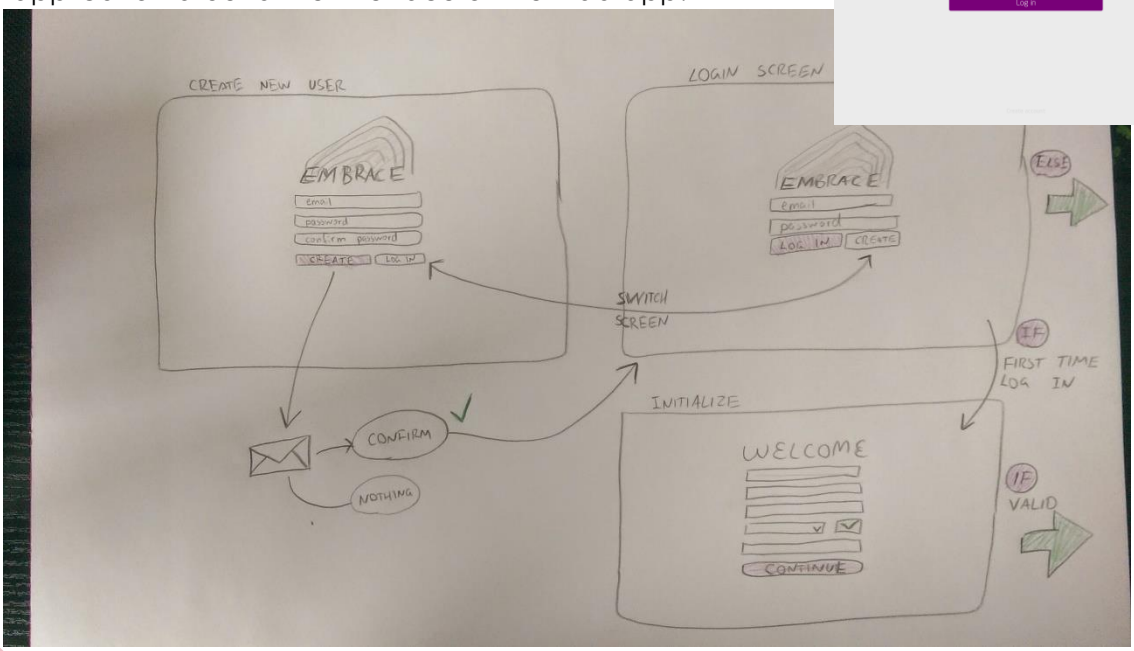
User Interface Specification

In order to help visualize the view, and make the implementation of the interface easier. This section will go through some of the visuals which should help streamline the implementation phase, due to the fact that it is not being designed "blindly".

Login

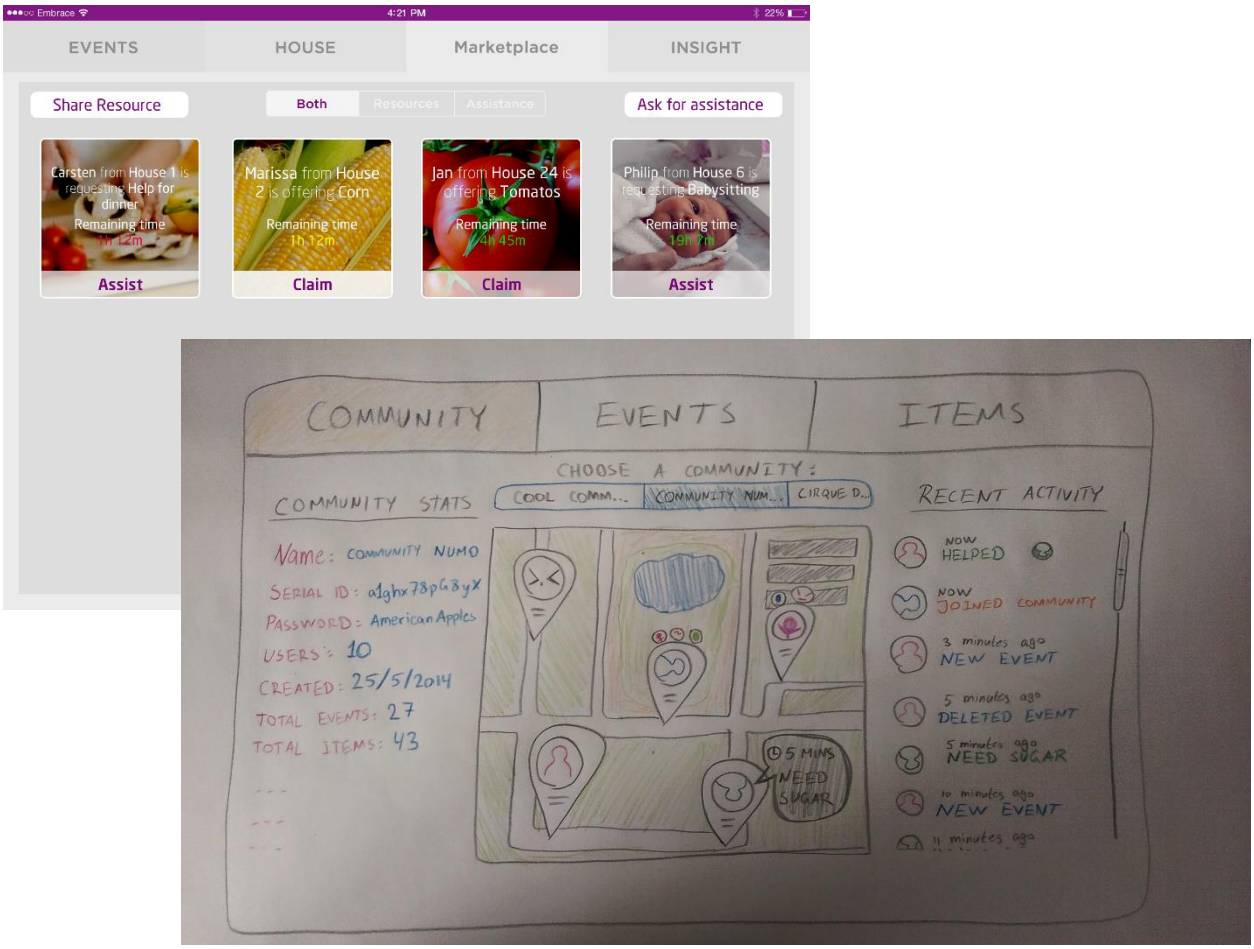
Due to the late decision of deciding to make the wep application as a standalone module, most of the time in the beginning was invested into the IPad app, and the EPIC cloud. Therefore the easiest way to get started, while not having wasted too much time during the early phases of the project, was to build the interface in the web application around the interface of the IPad app.





CRUD

How the user is supposed to be able to Create, Retrieve, Update and Destroy entities.



As seen here, the proposition is to somehow display the data in an interactive way, maybe even with cross-community support.

Chapter 4

Implementation

Upon finishing the design phase of the project, the project should start to take shape, and due to the decisions made early in the design phase, it should be less prone to unexpected time consuming features. This part of the report will present the actual product on a low level, and elaborate on how the requirements in the previous chapter was met.

Since one of the requirements was to have the platform available for as many devices as possible, an app was not a possible solution. One thing all devices have in common though, is access to the internet, and some sort of browser, which supports HTML, JavaScript and CSS. Therefore it was decided to create the platform, using these languages.

It will consist of four different components:

- ✓ Interface (HTML, JavaScript, Knockout.js, CSS)
- ✓ Router (JavaScript, Sammy.js)
- ✓ Server (PHP)
- ✓ Database (MySQL)

Authors Notes

The author, prior to this project, had no experience with...

- Knockout.js
- Sammy.js
- PHP
- MySQL

Each of these components will be linked according to the proposed architectural model mentioned in **Chapter 3: Design** on page 17. Each of these four components will be explained individually in the course of this chapter.

The Interface

The interface (or view) is part of the client. This means, that the router, are very closely connected, but for the scope of the presentation, they have been separated, since they have different dependencies, and have different roles to maintain.

The interface was developed with the markup language HTML, the scripting language JavaScript, the styling language CSS, an extension to JavaScript, JQuery, and another extension to JavaScript, Knockout, which enables the Model View Viewmodel design pattern.

Source Files

The files which are used in this part of the report are...

- index.php
- functions.js
- style.css
- knockout-3.1.0.js
- jquery-1.11.1.min.js

Displaying Objects

Since it has been developed as a single-page application, all elements have to be loaded somehow in the memory, and be hidden. This is done using the 'visible binding' in Knockout, which evaluates whether or not an element should be shown, depending on whether or not something evaluates to true or false. When the

elements are hidden (display:none), all sub-elements are still in the HTML DOM (Document Object Model), which means they can still be manipulated and accessed if the user wants to. But some things are not supposed to be accessed at certain times. For example: If you are not part of a community, you should not be able to create events, or items to put on the marketplace. Therefore when no communities are received in the view model upon logging in, the elements should not be placed in the DOM at all, this is done using the 'if binding'. As soon as a community is registered, Knockout (JavaScript) will put the proper elements into the DOM. In other words, when the objects in the viewmodel gets updated, the view will automatically be updated as well. The 'if binding' is very strong since it prevents users from accessing functions, which they should not be able to access at the time (e.g. trying to log in when already logged in).

An example of elements using the 'visible binding' are the menus. There are three menus, one for each class; Community, Events, Marketplace. As soon as you log in, you will receive all active events, communities and marketplace items. These will be loaded into the viewmodel, but depending on which tab you are currently on, the others will be hidden with the 'visible binding'. This is crucial compared to the 'if binding' because the elements can still be accessed since they are still in the DOM. This means that while being on the Marketplace tab, there will still be updates going on in the background to both the events and communities, so when you choose to switch tabs, everything will be loaded and ready, leaving the user with a very smooth experience.

In order to display objects from the viewmodel in the view, there are several different data bindings. One of the most used ones are the 'text binding', which simply prints the value of the variable in the element, so for example:

`<div data-bind="text:value"></div>`, where `value="I like cheese";`
= `<div data-bind="text:value">I like cheese</div>`

It is important to clarify, that the variables, which has to be used in the data bindings, has to be constructed using a Knockout function, so they are 'observable', basically what this means, is that it adds a lot of event listeners to each observable, and the objects in the DOM they are attached to.

Actually there are three pages, which can be showed, two of which are mutually exclusive.

- Login / Initialization (Mutually exclusive)
- Menus + Chosen menu content (Mutually exclusive)
- Loading (A .gif image, which says Loading)

There are three 'observables' controlling which pages to be shown. `Login`, `init` and `loading`. When loading gets set to true, no matter what is currently on the page it will be 'grayed out' and the .gif will start spinning, and prevent mouse interaction with other elements. This is done using the 'css binding', which allows for certain classes to be applied, when certain conditions are met.

Object Flow

Most of the times, there are more than one object, or the object contains several fields. In cases like these, it is necessary to dig inside the object/array. This is done using the 'foreach binding' or the 'with binding'. The 'foreach binding' creates a new object in the DOM for each element in the specified array. What is so unique about both of these bindings, is that they "dig" into the scope of the defined object. So for example:

```
abc={def="Some",ghi="Thing"};
<div data-bind="with:abc"><b data-bind="text:def">Some</b></div>
```

So originally the variable 'def' did not exist, but as soon as the 'with binding' were used, the context changed, and it was possible to access the children of the parent element 'abc'.

The 'foreach binding' is used for displaying several communities, as well as events and items on the marketplace. Each time a new item or event is created, and the viewmodel gets the information, then it will simply get added to the array of objects, and automatically be added to the view. On the other hand, when one of the objects are deleted or edited, the request will go to the model (server), and after accepting the change, the object simply gets removed from the array, and the view is updated accordingly.

Event Dates

In order to get as much info as possible on the screen initially, the dates for the events are quite long, since they write day name, date, month name and year for both the start and end of the event. Deep down in the programming clutter, there are some logic, which calculates whether or not the start or end date is today or tomorrow, and prints that instead of the long expression. So instead of "Monday the 25th of August 2014" it would say "Today". If the event starts and ends the same day, it could potentially print something along these lines: "8:30 till 12:31 Today (duration: 4 hours and 1 minute)"

House Addresses

Due to the fact, that several people should be able to live in the same house, there has to be some sort of way of normalizing addresses. No matter how the data was trimmed, or shaved down, there was still the possibility of spelling mistakes, and ambiguity (e.g. Main Road 27 could be everywhere in the world). The best fix was to introduce a search bar, which used the Google Maps API. This way it was possible for the user to choose a very specific address from a drop down menu. But just because there is a search option doesn't mean it will get used, therefore, just before the initialization form gets submitted, a Google Geocode query is sent. Basically geocoding is the possibility of sending either a query or a coordinate and get back geographical data. In this case, if the user types something ambiguous, the algorithm will simply choose the first suggestion from Google.

Limitations

The free Google Geocode API has two limitations…

- 2.500 requests/day
- 10 requests/second

This means that there can be 2.500 new users per day.

Design

The design was one of the challenges, especially seeing as multiple platforms were targeted. The issue is, that each browser has its own compatibility issues, and own parsers and engines, which means that some lines of CSS might work fine in Firefox, but crash and burn in Internet Explorer. Trying to keep it simple, on a functionality level was very hard, since so many functions comes with either JQuery or Knockout, which only works for certain browsers. At the same time keeping the interface simple with CSS was just as huge of a challenge, since some browsers react differently to the same CSS rules. There is a clear example of that, in the style sheet, where there is a special rule for Firefox (`@-moz-document url-prefix()`). This leads to another experience in Firefox, with the placement of the user portrait, but other than that, the use of relative percentages instead of absolute values have made the interface scalable to smaller resolutions, which enables the use on several devices.

> "Simplicity is the ultimate sophistication."
>
> **- Leonardo Da Vinci**

One of the important aspects of the design, was to use a lot of the preformatted markups in HTML, like forms and input with types. This is especially advantageous, when it comes to smartphones and tablets, where the browser opens special dialog boxes for certain input types, which makes it easier to fill out the forms (except if you use Internet Explorer, then you still have to type everything manually – luckily Microsoft tablets comes with keyboards these days…).

The Router

In order to track the history, and go 'back' and 'forward' in browsers, there has to be some sort of URL change. Besides, since the design of the website is a single-page application, page reloads are very harmful (since they clear the data), so there has to be something to handle the URL changes, without reloading the pages. This is where the router comes into the picture. Basically the router is a combination of the built-in JavaScript function location.hash, and the JavaScript library Sammy.js. Basically the website, being named index.php is accessible from just the root folder of the domain, but then upon running the router the URL is redirected to match the current state of the application.

Source Files

The files which are used in this part of the report are…

- functions.js
- sammy.js
- jquery-1.11.1.min.js

URL Matching

Basically, what Sammy.js is used for in the project, is that it adds listeners to the URL, and if the URL matches a specific pattern, the defined Sammy function is called. An example of a pattern could be `'#:page/:subpage'`. Basically this function would be called every time the hash of the URL switches to http://fakeurl.org#var1/var2, where `var1` and `var2` can be any valid URL syntax. You can then access the values of `var1` and `var2`, by using `':page'` and `':subpage'` from the pattern. They can be accessed using the Sammy function `this.params.[name]`, and then depending on the values, it can be handled in different

ways. It is also possible to make the patterns more specific, so for example `'#Login'` only reacts if the user specifically gets to http://fakeurl.org#Login. One of the patterns, which are used in the project are the `'#:menu/new'` pattern. The reason it has good value, is due to the fact, that each menu (Community, Events and Marketplace) has the option to open a form, and create a new object. This cuts down on redundancy. All of the forms used on the submit, actually submits to the `#:menu/new` page, which, upon being called when you are already on the page, will close the form, and redirect back to the menu URL.

Login

Due to the nature of the single-page application, everything will be cleared from memory upon doing a page refresh, and of course that is a minor problem, but essentially everyone has been trained for the last decade, that when they are logged in on a page, and they refresh, they stay logged in. Therefore it was necessary to somehow make sure they would be able to log in automatically upon a page refresh. The only logical approach was to place a cookie, which contained the credentials. It was quickly decided not to have the username and password stored, since that could lead to abuse, so instead upon logging in, the server returns your user ID, as well as a randomly generated unique token, which is stored in the database. This token is unique for the user's current session, and will not be valid after 30 minutes of being inactive. This means, that the user cannot be logged on simultaneously at different machines, or for that matter log in after closing the browser, since the cookie is scheduled for deletion upon the browser closing.

So the first thing that happens when a user enters the site, is that all variables are reset to their standard values, which means nothing is loaded. The next thing then is that it searches for a cookie, if no cookie is available from the domain, or it does not contain an id and a token, it will always redirect to the `'#Login'` page, but if there is a cookie, it will try to perform a cookie login. Depending on the outcome of the login, it either goes to the `'#Login'` page, or it goes to URL, which was typed when entering the site.

States

This is referred to as the `loginState`, so basically what this means, is that just before you get redirected, the URL hash will be remembered, and applied once you log in, the only exception being, if you need to go through the initialization progress first. Actually, there are several different variables keeping track of the states. One for each menu. This means, that when you go from one menu to another, it will remember the last URL in that menu. So if a user is trying to create a new event, but has to see something in the Marketplace, he can go to the Marketplace using the navigation panel, read what needs to be read, and then to go back there is two choices, 1) use the 'back' button 2) click the Events tab, since it will redirect him to the URL last visited, which in this case was the `Event/new`.

The Server

Source Files

The files which are used in this part of the report are...

- index.php
- embrace.php

Where the previous parts of the report described the frontend, it is now time to describe the backend. The server is basically the link between the client and the database, making sure, that nothing malicious are going to enter the system, or for that matter make sure no one can break in without the proper authorization. The server is really just one long file, with a lot of functions and switch/cases, where it decides how to respond to the request.

The Protocol

> "A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools."
>
> **- Douglas Adams, author of 'The Hitchhiker's Guide to the Galaxy'**

Basically the server follows a very strict protocol, which is there for security purposes. One of the things that is common for all the functions is that they assume, that the user can type in anything they want, and therefore every request should initially be considered a threat. This is also why the random unique token is used for authentication instead of the username and password, since those are static, but the token is dynamic, and changes each time the user logs in. Since there are no state tracking on the server, it can always run everything, as long as the credentials are correct. Each time a user sends a request to the server, a timestamp will be updated, in order to reflect when he was last active. The server will always encode the results in JSON format. If the request was successful, the server will respond with the variable `msg`, but if there is an error, it will respond with the variable `err`. Each of those containing a more detailed response.

Create New User

Seeing as the users has to use their email upon creation, it was very important to make sure, that it was a real email and not a fake one, or someone creating the account with someone else's mail (thus making them unable to create a user). Ultimately, when the user creates their account, their status is set to 'mail', and a unique token is generated, this token is then sent in a link to the user's mail. There are two links though, one for accepting, and one for declining. Basically the one for accepting will validate the user, by matching the unique token in the database, and see if the status is currently set to 'mail', if that is the case, their status is instead set to 'init', which means they have to go through the initialization step when they log in next. This also means, that the link will no longer have any effect, since the status has irreversibly been changed away from 'mail'. The link for declining will instead remove the user from the database, and should put a warning flag on the IP address, which submitted the email (a lot of flags = IP blacklist). The latter function has not been implemented yet. The links from the mail sends the user to the index.php page, and not embrace.php.

Login

There are two types of login, the cookie login, where the server expects an id and a token, or the regular login, where it expects a username and a password. Upon confirmation, the server returns the data, which are currently active for the user logging in.

CRUD

Each CRUD command has been implemented on the server, which means it is possible to Create, Retrieve, Update and Destroy both events, items and communities.

Community

In order to reduce computation a lot of information are in the databases. One of which are the bounds of the community, since that could be used for visualization of the community on a map. Whenever a person joins or leaves a community, the coordinates from the house which the person lives in, has to either be added to the bounding box (which consists of 4 values, north, west, east and south) or removed. When joining, it is checked whether or not the new latitude and longitude coordinates are outside the current box, and update the new bounds accordingly. Besides, when joining a community, all community data has to be fetched as well (events, items, neighbors etc.). Leaving is described in the following:

Algorithm 1 Leaving a Community

```
IF user is logged in THEN
    GET users, bounds and id FROM Community;
    IF user is linked to the community THEN
        GET house·coord, user·communities FROM User;
        IF community is linked to the user THEN
            Remove the community from the user;
            IF no users left in the community THEN
                DELETE the community;
                UPDATE the user, where community is removed;
                RETURN success TO CLIENT;
            ELSE
                IF house coordinates are on the outskirts of community bounds THEN
                    CHECK = TRUE;
                Remove the user from the community;
                FOR EACH user in the community
                    IF CHECK is TRUE THEN
                        Calculate the new bounds;
                UPDATE all the new values in the database;
                RETURN success TO CLIENT;
RETURN error TO CLIENT;
```

Worst-case run time $O(n+m)$ where $n = total\ communities$ and $m = total\ users$

The whole approach of the algorithm, is to do as little as possible. The worst-case run time of the algorithm is linear, where the total number of communities and users are the upper bound. These will be fetched through MySQL, which is optimized for these operations.

Items
Upon claiming an item, it is important to subtract the claimed value from the item, and upon doing that, write the user id, and the claimed amount in a log. This way, the person offering gets to see who has claimed, and the other users can see who to ask, if they REALLY need that item, which were on the marketplace.

Connecting to the Cloud

A very important aspect of the product, is that it should be able to integrate with the cloud solution, which was designed for the EMBRACE houses. During the login and cookie login, there is a special clause stating, that if the status is either 'embrace' or 'vacation' (which are the keywords used for the EMBRACE houses in the system), it should call a special function in addition to the others. This function will make sure to validate the user, with the necessary credentials in the cloud system, and each time the user requests for updates, they will also receive updates from the EPIC cloud. This is done through the built in command in PHP called CURL, which sends an XMLHttpRequest to a server (post/get).

The Database

The database contains the necessary fields in order for the model to be maintained. All of the database tables and columns can be found in **Appendix A: MySQL Tables** on page 48.

The Model

The model is a combination of MySQL and how PHP parses and delivers the input, but the rough sketch of the model is observable directly from the database. Some of the fields, which haven't already been defined in the design chapter, are some of the functional fields. Some of them have already been defined through the course of this chapter, but here is a brief overview of the programmatically functional data, and what it is used for:

ITEM

- TAKERS: History field, so they can see who has claimed the items.
- IMG: Contains a URL to an image, which will be showed in the interface with the item.
- DELETE and DELETE_DATE: There are no instant deletion of events or items, except if they are accessed from an archive and deleted again. This is good for undoing.

USER

- TOKEN: Used for security purposes, for unique sessions and the cookie login.
- GAME: True/false, in or out. Even though the data for having a game, will not be available on this platform, it is good to know whether or not to expect game data, when communicating with the EPIC cloud.
- STATUS: In order to determine whether the user has an EMBRACE house, is on vacation, is currently waiting for email authentication or needs to initialize their account.
- PHOTO: A profile picture of the user. If none is chosen, there is a standard value.

- LAST_ACTIVE: For security purposes. 30 minutes of inactive = logout.

EVENT

- HOUSE_AUTH: If creating an event in someone else's house, this will be 0, and it will not be 1 before they accept the event. All others than the host and the house owner will not be able to see events until the value is 1.
- DELETE and DELETE_DATE: The same as in Item.

HOME

- COORD: The coordinates of the house address, which are fetched through the Google Maps API – they should be used for visualizing the locations of users in the community.
- EMBRACEID: If the house is an EMBRACE house, the ID for authentication on the EPIC cloud should be defined here.
- POSITION: The possibility of rotating or flipping the house model, which are in type.

COMMUNITY

- SERIAL: A unique serial ID, which is used for identifying the community so others can join it.
- BOUNDS: This is a computed value, from all the house coordinates from the houses in the community, which displays the square in which all houses will be contained. This is supposed to be used for visualizing the communities on a map, and making it a more interactive experience.

MySQL Commands

In order to Create, Retrieve, Update and Destroy, there are a few basic MySQL commands. Create = Insert Into, Retrieve = Select, Update = Update, Destroy = Delete. These are the basic commands used for manipulating with data in tables. The idea is to connect to a database, and construct queries, which works much like a request, where MySQL will then respond with what happened. So for example, it is possible to retrieve all data from a community where the credentials is known using the following query in PHP:

```
$q ="SELECT * FROM Community WHERE serial='$sr' AND password='$pw'";
$mysql->query($q);
```

This seems rather simple, and if the input is trusted, then it can do no harm. But if the input can come from anyone with an internet connection, then it opens up for trouble.

SQL Injections

If the input is directly put into a query, there is the risk of having SQL Injections. For example, if the input is a string like this: `$sr="String'; DROP Table;'"`, basically what could happen here, is that the query could look something like this: `"SELECT * FROM 'Table' WHERE serial='$sr'"`. This would be bad news, because really, the string will look like this: `"SELECT * FROM 'Table' WHERE serial='String'; DROP Table;''"`, which basically means, that it will initially match serial with `'String'` and fetch the proper rows, but

afterwards it will do bad things. Of course, this example is rather primitive, but in some SQL versions it would still work.

One of the measures to guard against these attacks, is by using parameterized queries. This is a special type of PHP syntax for sending queries to MySQL, which requires the input to be defined as either a string, integer, float etc. This means, that the input is handled in a proper way and sanitized before getting entered. A parameterized query could look like this:

```
$mysql->prepare("SELECT * FROM 'Table' WHERE id=? AND str=?");
$mysql->bind_param('is',$id,$str);
$mysql->execute();
```

Basically this would expect an integer where the first question mark is placed, and a string where the other one is placed. Originally the intention of parameterized queries, was to improve the run time, if there was a need to loop the same string over and over again. There would only be the need to bind the new parameters, and therefore, the time parsing the rest of the query would only be counted once, and in the long run, that would make a difference. Recently it has instead become a way of sanitizing input.

Compatibility

In order to run all of these modules, you need something which can compile and run the code. Each of the modules have different requirements and versions.

According to the http://PHP.net manual, prepared statements with MySQLi, requires "version 4.1.13 or newer, or 5.0.7 or newer" of MySQL, and requires "PHP version 5.0.0". Both of these were released prior to 2006 (PHP 5.0 was 10 years ago). No other functions in the PHP files are younger.

For the JavaScript libraries, Knockout is the best supported, since it runs back to Internet Explorer version 6. Sammy.js states that they should work with all the browsers, which JQuery works for. Even though there is a JQuery version 2, the version which are being used in the scope of this assignment is 1.11.1, which according to http://jquery.com/browser-support goes back to Internet Explorer 6, the current and last version of both Chrome, Firefox and Opera. It also works for latest version and previous major version of mobile Safari (iOS) as well as Android 2.3+, which was released 3 years ago.

So far the core features are supported by all the major browsers, and running back to IE6 is quite an achievement. The Google geocode also works in IE6, but if Google Maps (an actual map frame on the screen) were to be implemented, the support would fall down to IE8 according to https://developers.google.com/maps/faq#browsersupport. It is important to note, that it is over 5 years ago IE8 was released.

Overall the support is good – the only troublemaker are the CSS interpreters in the browsers.

Chapter 5

Results

"The project went exactly as I expected!!!" – said no one… EVER! This part of the report will display the results, and cover all of the sides, which are relevant to the problems laid out in the introduction.

The Website

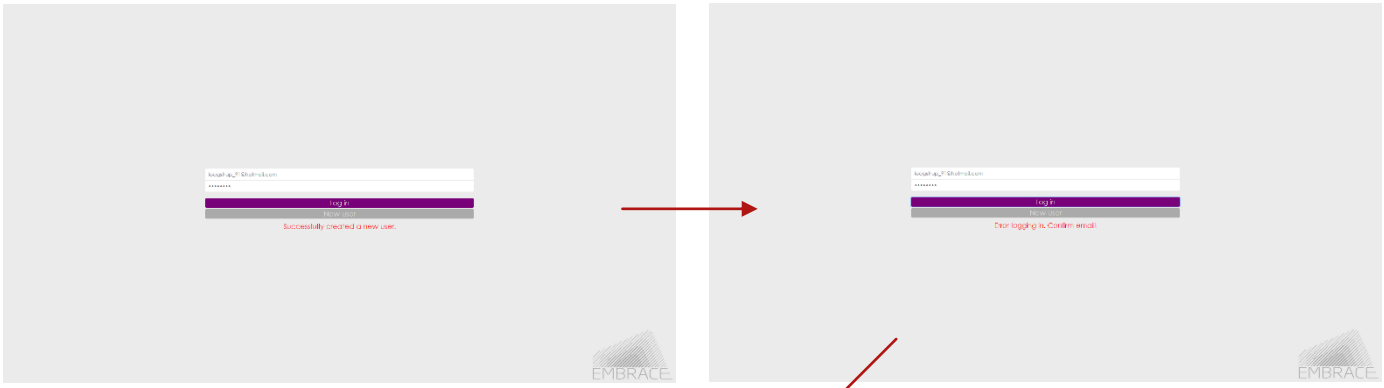
What is the best way to present a website? Well there is an old saying: "A picture says more, than a thousand words.", and since 10.000 words might seem a bit out of hand, the following part will primarily consist of screenshots taken from different devices, in order to display the design as well as functionality.

Authors Notes

You can check out the live version of the website at… http://syv-it.dk/PO/embrace

Desktop

Since the desktop are the most tested version, the main focus here will be to portray the implemented functions.



Patrick Olesen
til Patrick Olesen

E-mail Verification

Welcome to EMBRACE

Please click the following link, in order to complete the registration.
Validate your e-mail

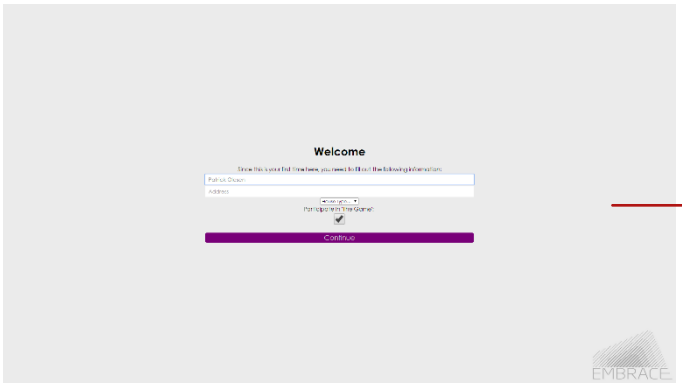
After registering, please log in with your credentials:

E-mail: loegstrup_91@hotmail.com
Password: Guest123

If it was not you who created this user, please click the following link, in order for us to prevent the person from harassing others:
Remove your e-mail from the system

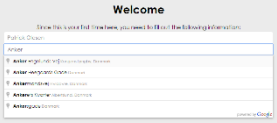
Authors Notes

Here the user creation process is illustrated.



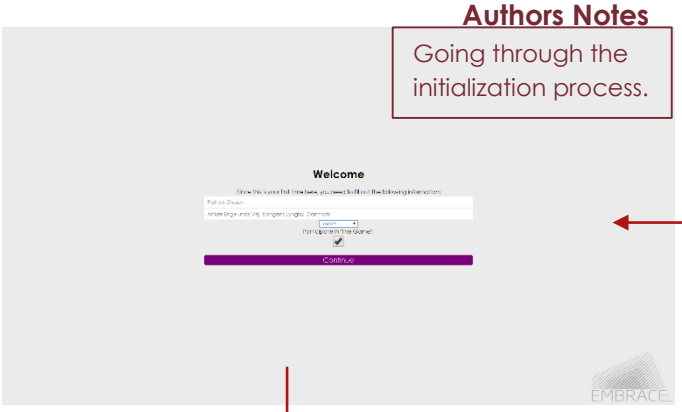
Authors Notes

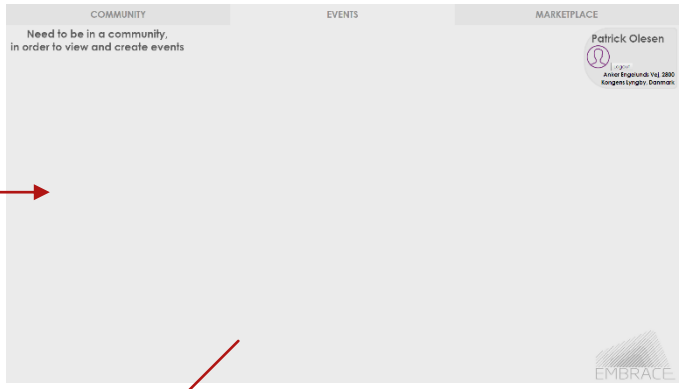
Notice the Google search box to avoid ambiguity.

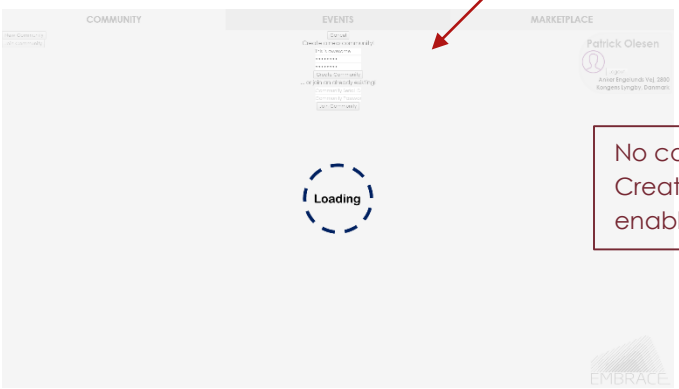


Authors Notes

Going through the initialization process.

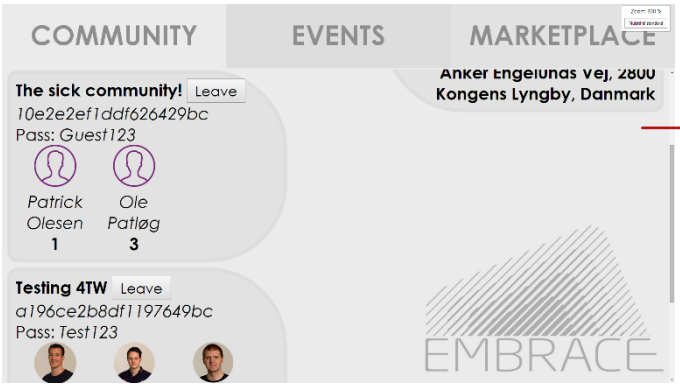


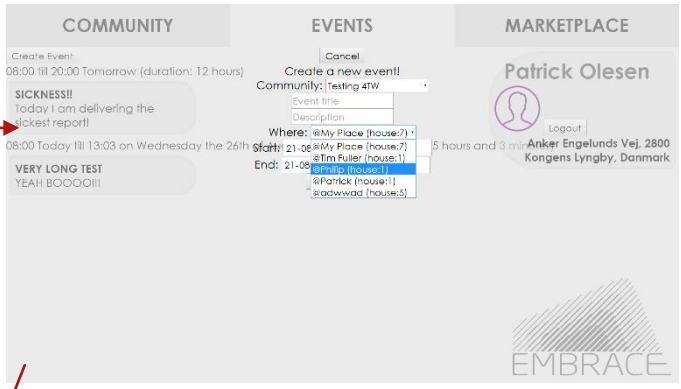




Authors Notes

No community, no fun! Creating one in order to enable events and items.



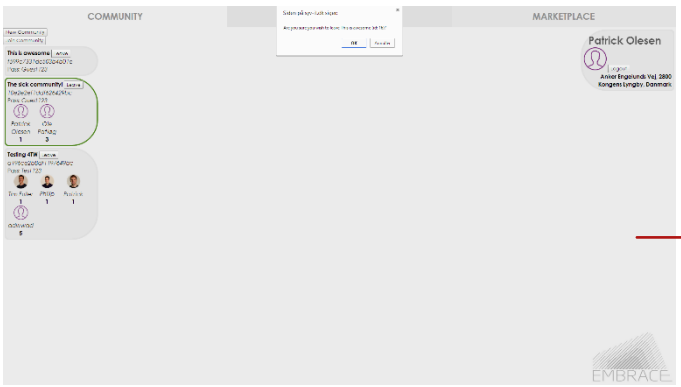


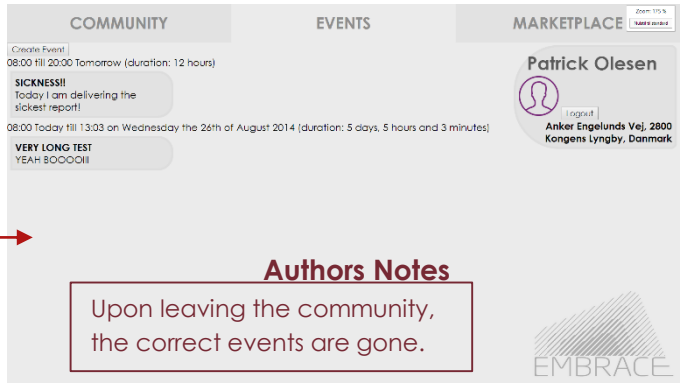
Authors Notes

Zooming in, and nothing broke. Minor miracle…

Authors Notes

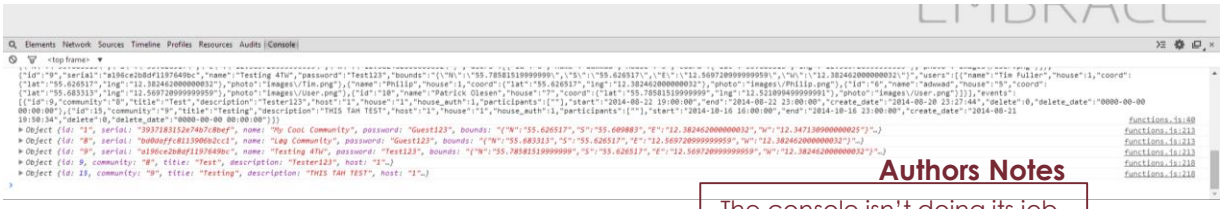
Notice how the options adapts to the chosen community.





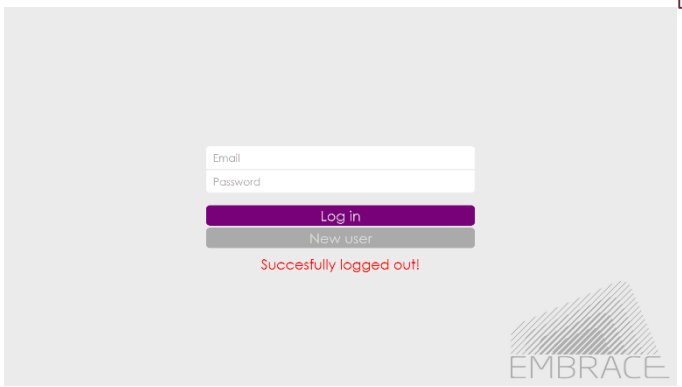
Authors Notes

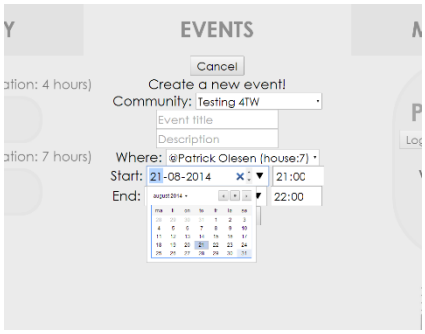
Upon leaving the community, the correct events are gone.



Authors Notes

The console isn't doing its job right if it isn't spamming you!



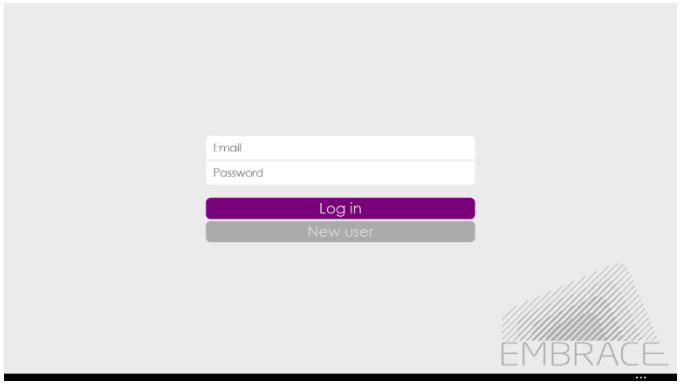


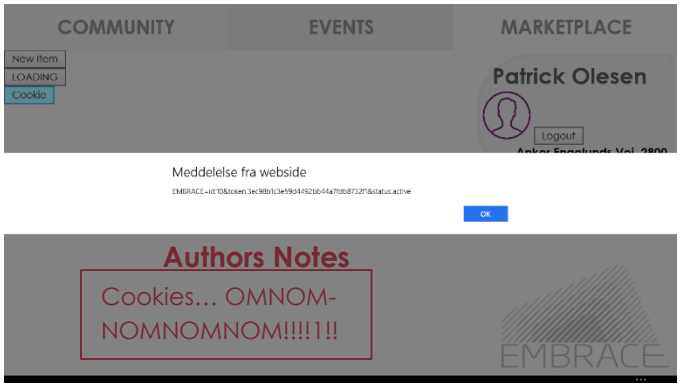
Tablet

Taking some quick screenshots quickly ends up with having to make bug fixes. Since all the functions have already been showcased, this is just to show how it fits to the resolution. The tests are on a Microsoft Surface, running Internet Explorer.

Authors Notes

Actually a very fitting standard resolution.





Authors Notes

Just when you appreciate Internet Explorer the most, this happens...[7]

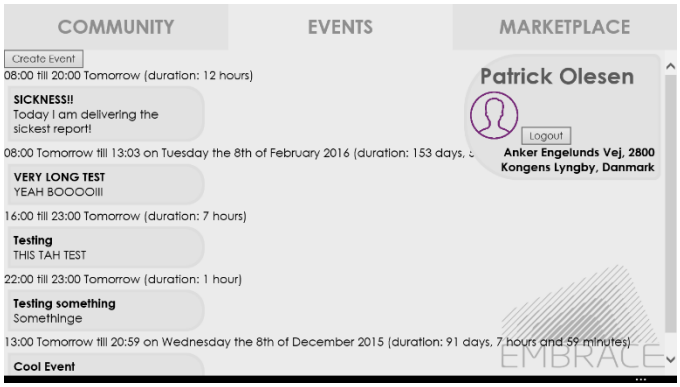


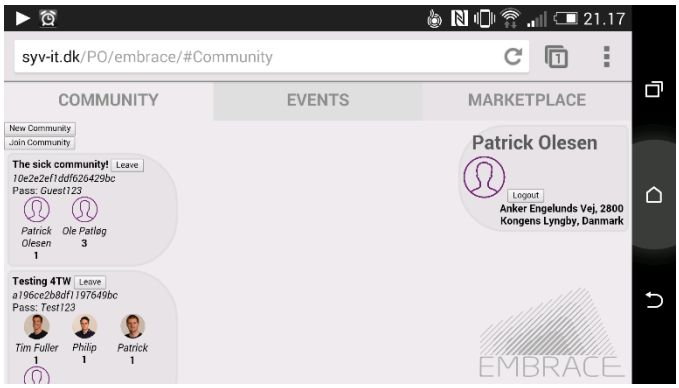
Figure 12 – Fixed the bug, using the procedure in the link... Oh wait... 153 days... Back to the drawing board!

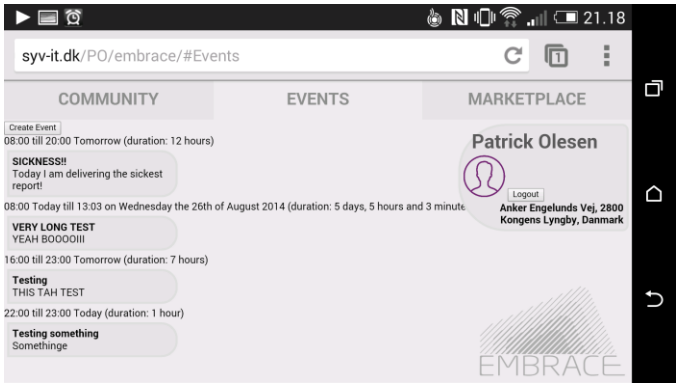
Figure 13 – Here the bug is actually fixed.

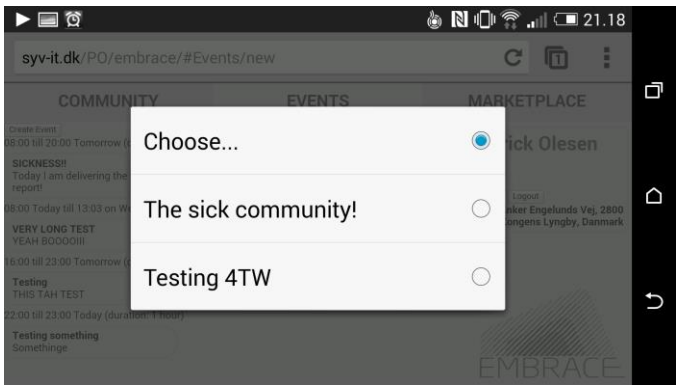
[7] http://biostall.com/javascript-new-date-returning-nan-in-ie-or-invalid-date-in-safari

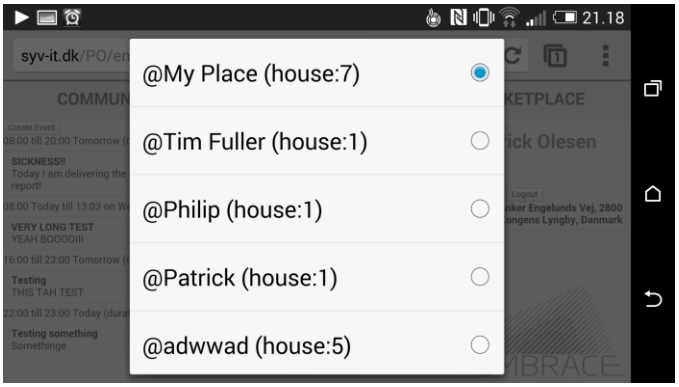
Mobile

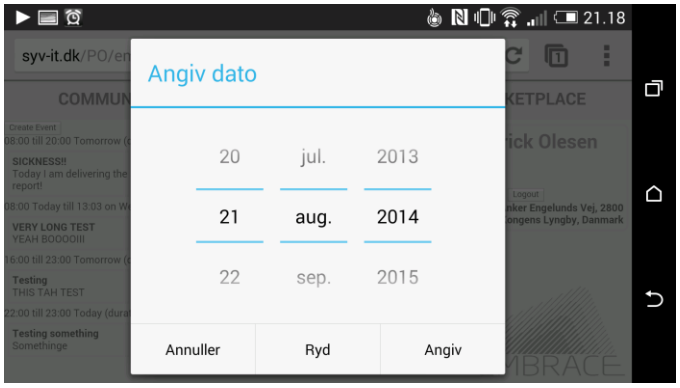
The mobile which was tested on, is an HTC One (M8), in the Google Chrome mobile browser. The focus here, is on the mobile input boxes.

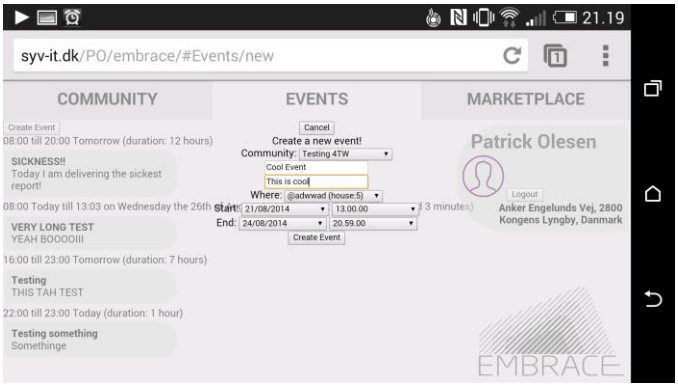


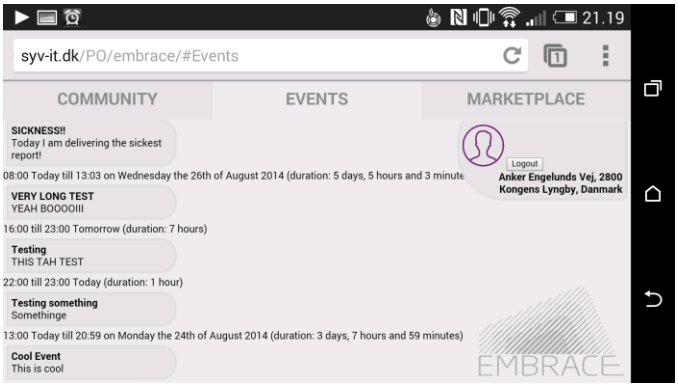












Authors Notes

Not much to say. Everything looks fine. Especially the input fields. All libraries work, which can also be seen in the URL.

Chapter 6

Evaluation

This part of the report is for evaluating the project, and to talk about what are missing, what have been done, what haven't been done and why. Besides it will look forward, and try to say something constructive about the future of the product.

The Goals

The first thing to do, is to summarize the goals:

- ✓ Describe and present a software system
- ▪ Engage residents in small neighborhoods to share resources
- ✓ Engage them to socialize and know neighbors better
- ✓ Available to every type of household, not limited to EMBRACE
- ▪ Integrate with the cloud
- ✓ Form its own identity compared to competitors
- ▪ Make it responsive and easy to use
- ▪ Test in terms of memory and speed

The Result

The points which have been sufficiently done, is marked with a check sign, the others will be elaborated on in the following part of the report.

Check ✓

The product which has been presented have been well described through the course of the report, and the functionalities, which have been implemented, have been tested on different devices, where they have been proven to work.

These functionalities are user creation, mail authentication, login, initialization, CRUD commands for the community, create events and logout. It is important to note, that every single CRUD command for both events, items and communities have been made on the server (two things are missing, you cannot leave an event, or undo claiming an item), and it is only on the client side, where they still need to be implemented.

Availability wise, the design has really paid off. Due to the flexibility of HTML and JavaScript, it allows for the application to be launched from all kinds of devices, which ultimately means it is available to all kinds of households.

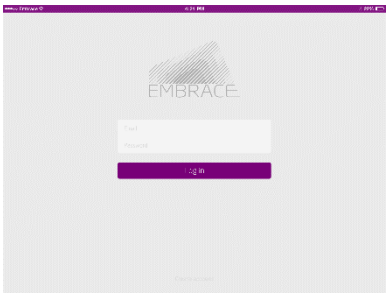
Identity wise, the project has its own unique style. It is not really bringing anything new to the table, seeing as there is already the option to make events and share resources on Facebook, but the concept of having a private community, which focuses on open communication, trust and actually only communicating in person (since there are no chat… by design!), it goes for a more practical approach. Of course the lack of functions implemented in the product makes selling it hard, but the design compensates for now.

Client

As mentioned a lot of functions are currently missing from the client. Since the client is what is visible to the users, this hurts a lot of the presentation value of the product. Being able to compute stuff on a server doesn't really sell that well. Especially when the server is depending on the client to feed it the input. The functions that have been implemented so far work very well though. Even some of the very small details, which no one notices when it works, but is weird if it doesn't. An example of that, is the time input fields in the create event form. When something is typed in the start time, it will automatically update the end time, to one hour after. If the end time is 4 hours after the start, and the start time is edited, the end time will automatically be updated, to be 4 hours later.

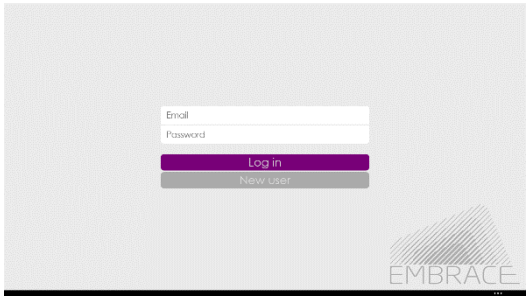
Cloud integration

As it is right now, everything is ready to integrate the cloud. It checks at the right times whether or not to update the cloud (and only if the user is actually residing in an EMBRACE house), as well, but due to the cloud API not being fully developed, there are no chance of testing or integrating with the EMBRACE houses.



Design

Regarding the design it did work on multiple devices, so in a sense the design is responsive. The issue though, is that it does not fulfill the defined design criteria. A lot of widths are fixed, and they WILL overlap, if the viewport is small enough. But other than that, the design has successfully maintained the feel and look, which was described in the design chapter. One of the positive notes, is that the design is intuitive, which means that there are no hidden things, and stuff which are hard to find. That means that the learning curve is close to non-existant.



Testing

The amount of testing (at least the kind, which goes in a report), has been very limited. Seeing as the server is the most established part of the framework, tests would have been essential in order to establish that claim. Instead time has been used in an attempt to catch up on the client side. There are some positive things worth mentioning though. The testing of the user interface for one are well documented, and actually there is a list of known bugs, attached in the source files.

The Future

So what are the next steps moving forward? Are there any problems, and what can be done to solve those?

Large-scale

> "To *improve* is to **change**.
> To be *perfect* is to **change often**."
>
> **- Sir Winston Churchill, Former Prime Minister of UK**

One of the things, which are worth talking about, is if the framework goes live and becomes very popular. A lot of interesting problems could arise there. First of all, a lot of people, all over the world, would connect to the server, so the question is, will it be able to handle it? Well judging from the code analysis, as well as the chosen design pattern, the real bottleneck lies in the updates, which are issued every 60th second. The reason this is an issue, is because most of the things which are sent back, have already been received before. This can be effectively solved using signaling. That claim will be further explained later.

Another issue could be the fact, that the server service, is a rather huge file – containing over 1000 lines of PHP code. Servers today are pretty clever. If a piece of code is accessed often, it will be cached, and available very quickly. This is why the first time you open some websites takes forever, but once you are one it, the template is cached. These automatic measures are a good solution, but the best solution could be, to split the file in several smaller files, which all includes the same header (some of the common functions). This way, the overhead is way less, and the loading times should be substantially better.

Memory wise, the database would quickly be filled up, but the idea is, that there is going to be a cleanup function, which runs every time someone logs out, which cleans up data, which have been expired over 30 days ago. Of course the archive won't be as fine, but the runtimes of the system will make up for it.

One more thing, which is a huge help is again the chosen design pattern. Due to the asynchronous nature of JavaScript and the choice of making it as a single-page application, the interface will still be usable even if it is waiting for a server response. Or at least it is intended to work that way. So when creating a new event, a loading GIF will not block the page, but instead be running on the new event button, and as soon as the response comes back from the server, the event is automatically added to the view, and then you can access the button again in order to create another event. So even while waiting, the user will not just stare at a white page, and the worlds slowest loading bar.

Database Optimization

Another way of preparing the system for going large-scale is by using effective queries and data fields. Upon finishing the server, it became clear, that there are ways of optimizing the MySQL queries, and some of the data in the tables. For example, when searching for a user right now, the comma separated list is retrieved, then analyzed in PHP and then SQL is accessed again. Instead there are some basic string functions, which allows for searching directly in MySQL in comma separated lists. At the same time, it became clear, that the time comparisons in MySQL with datetime are slow, compared to having UNIX timestamps.

Functionality

Nice-to-have

Interactive Map
As previously mentioned, having an interactive map showing could be very awesome. One of the key reasons being, that it adds something unique to the design. The concept is, to use the boundaries, which have been calculated and stored in the databases, to display the residents of the community on Google Maps, where each house will be marked with a pin. These pins will then be able to put out chat bubbles and other notifications, which alerts the user, if something is happening at their place, or a new item/event was just created by them. Having it shown on a map instead of text, will give a much more personal feel.

Another possibility is having another view option, where it displays the neighbors where you have placed them, and the element they have been placed on, can be an image, which the user has uploaded. This is highly relevant for people living in apartments or dorms, since the pins would most likely overlap on Google Maps. That way you can draw the apartment complex from the side, and place the neighbors in their respective apartments.

Effective Updates (Signaling)
Another really nice thing to have, is signaling. What is meant by this, is that each time something new is submitted to the server, it marks it as unread for everyone, and sets a signaling flag for each user, that something new has happened. That way, the next time when they log on, or requests an update, the server will just search for all the things which are unread, and post those to the user. This way, the user will not receive the same data over and over again, except of course if they refresh the page somehow. This allows for having more frequent updates, since the traffic has been reduced a lot.

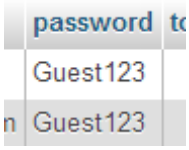
Gamification
Yes, this is indeed very nice to have. Even though the concept would work best if there was data from their house, there is still ways of applying gamification to the non-EMBRACE houses. One of the ways could be to add hints, tips and challenges to the system, which does not keep track of whether or not the things are being done, but just for the individual to on his/her own. The good thing about introducing this, is that it also works as conversation starters with the neighbors, and raising awareness about the environment is never a bad thing. Another way is by having messages pop up, when an event is over, which could be like: "Congratulations, you were just 5 people @Insert Name, based on average measures, you just saved XXX kilowatt, which corresponds to XXX kr."

Need-to-have

Client Functions
Of course the most necessary thing which needs to be implemented are the client side functions. These things have already been mentioned and until they have all been implemented, the system is not usable in a daily context.

Encryption

The fact remains, that a lot of data which comes into the system, are sensitive data, which are not supposed to be read by anyone or anybody. Therefore there needs to be some sort of encryption.



It doesn't really matter how the data is encrypted, as long as it gets smashed beyond recognition. In order to be able to match passwords, the most intuitive way is to make the encryption deterministic. This does open up for the possibility of decrypting it again though. If a user forgets their password, then you simply put it up for deletion, and send them a mail, where they have to verify it was them who scheduled the password change. There is always the possibility of brute-force attacks, but one of the things which have been done to guard against that, is that if the user enters a wrong password, there will be a 2 second delay, before the server responds. This leaves a maximum of 30 attempts per minute. Of course there are ways to bypass this (having ton of browsers open), but the idea is good.

Authors Note

The course of this project has been strongly dominated by the lack of time. Throughout the first 2 months of the project, all of us worked on the architecture and design of the cloud system, and it wasn't until June (actually the week where I were in Versailles participating in the competition), I decided that I should make a standalone framework. This decision was made partly because I hadn't managed to find my identity in the cloud system, and simultaneously, because the person responsible for the app was unable to integrate the events and marketplace into the app anyway in time for the competition.

So considering the fact that most of my work being presented in this report have been done over the course of 2 months, where I have managed to learn both Knockout (MVVM), SammyJS (Router), PHP and MySQL, I think it has turned out alright. Some of the things which are not possible to express in the report, since it is irrelevant to the product, are the experiences I have made, both in terms of Solar Decathlon, which taught me a lot about intelligent houses, but also in terms of what I have learned in regards to the .NET framework while working on the cloud solution.

Chapter 7

Conclusion

Waste of resources, loneliness and environmental difficulties. Those were the problems, which this report set out to investigate and try to help improve. What all three of them have in common, is that there is no quick fix for either of them.

The suggested solution was to implement a new kind of social network. A social network, which should be specifically designed to work in local communities, trying to enable the users to socialize and share resources while raising awareness about the environment.

Even though the original concept was to only target the social network at people living in the so called EMBRACE houses, the concept was later expanded, to also include all other kinds of people with an open internet connection.

In order to make the framework wide accessible, it was decided to develop it in HTML, JavaScript and PHP, which is accessible from pretty much all browsers. Furthermore it was decided to design the website as a single-page application, due to the many advantages.

Since the use of tablets and smartphones have increased dramatically over the last five years, it was also decided to go for a responsive design of the application. This means that the website should run smoothly, and fit onto all screens, no matter what size the viewport has.

Even though a lot of the goals were reached, the overall product is not satisfying. The reason being the lack of essential functions in the client, which were necessary in order to both test properly, and also in general to make the application appealing. It is not all bad though, since the server has been fully developed, but from a presentation point of view, the client is definitely the most essential part.

Moving forward the framework definitely have potential. It manages to stand out compared to all other major social networks, since it focuses on social interaction and communicating face-to-face rather than being tied at the computer. This is not going to solve all the world problems, but it definitely has the potential of improving life quality by urging people to spend more time together. It will especially be good for the EMBRACE houses, when the cloud solution becomes available.

Overall it has been a good learning experience and a valuable lesson. Both in terms of learning some new concepts, design models and languages, but also in terms of being better at distributing time and energy.

References

[1] P. Thibodeau, "Computer World," 8 march 2013. [Online]. Available: http://www.computerworld.com/s/article/9237459/Computer_science_enrollments_soared_last_year_rising_30_. [Accessed 18 august 2014].

[2] Danish Education and Science Ministry, "Enrollment Statistics (Danish)," 15 july 2014. [Online]. Available: http://ufm.dk/uddannelse-og-institutioner/statistik-og-analyser/sogning-og-optag-pa-videregaende-uddannelser/grundtal-om-sogning-og-optag/kot-hovedtal.

[3] Miniwatts Marketing Group, "Internet World Stats," 11 august 2014. [Online]. Available: http://www.internetworldstats.com/stats.htm. [Accessed 18 august 2014].

[4] Danske Medier, "FDIM," 2012. [Online]. Available: http://www.fdim.dk/sites/default/files/mediearkiv/rapporter/danskernes_brug_af_internettet_2012_rapport.pdf. [Accessed 18 august 2014].

[5] Pew Research Center, "Pew Research Center," 2014. [Online]. Available: http://www.pewinternet.org/fact-sheets/social-networking-fact-sheet/. [Accessed 18 august 2014].

[6] P. Parigi and W. I. Henson, "Social Isolation in America," *Annual Review of Sociology,* vol. 40, no. 1, pp. 161-166, 6 march 2014.

[7] S. Olsen, "New York Times Blog," New York Times, 5 november 2009. [Online]. Available: http://bits.blogs.nytimes.com/2009/11/05/does-technology-reduce-social-isolation/?_php=true&_type=blogs&_r=0. [Accessed 14 august 2014].

[8] Z. Hutson and S. Gamble, "West Virginia University Extension Service," october 2007. [Online]. Available: http://www.wvu.edu/~exten/infores/pubs/fypubs/WL_302%20Know%20Your%20Neighbors%20Leader.pdf. [Accessed 15 august 2014].

[9] Daily Mail, "Daily Mail UK," Daily Mail, 14 october 2013. [Online]. Available: http://www.dailymail.co.uk/news/article-2458396/A-recognise-neighbours-Poll-finds-wouldnt-able-pick-line-up.html. [Accessed 14 august 2014].

[10] Københavns Kommune, "Københavns Kommune (The Copenhagen Region)," march 2014. [Online]. Available: http://www.kk.dk/~/media/3444B8D6409A48A1BBB991106736E243.ashx. [Accessed 16 august 2014].

[11] C. Miller, "Food Waste," *Waste Age,* vol. 44, no. 9, p. 52, october 2013.

[12] Stop Spild af Mad, "Stop Spild af Mad," [Online]. Available: http://www.stopspildafmad.dk/madspildital.html. [Accessed 15 august 2014].

[13] J. Heggestuen, "Business Insider," 15 december 2013. [Online]. Available: http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10. [Accessed 15 august 2014].

Appendix A

MySQL Tables

Overall

	Tabel	Handling	Rækker	Datatype	Kollation (Collation)	Størrelse	Overhead
☐	Community	Vis Struktur Søg Indsæt Tøm Slet	4	InnoDB	utf8_danish_ci	16,0 KiB	-
☐	Event	Vis Struktur Søg Indsæt Tøm Slet	12	InnoDB	utf8_danish_ci	16,0 KiB	-
☐	Home	Vis Struktur Søg Indsæt Tøm Slet	4	InnoDB	utf8_danish_ci	16,0 KiB	-
☐	Item	Vis Struktur Søg Indsæt Tøm Slet	0	InnoDB	utf8_danish_ci	16,0 KiB	-
☐	User	Vis Struktur Søg Indsæt Tøm Slet	7	InnoDB	utf8_danish_ci	16,0 KiB	-
	5 tabeller	Sum	27	InnoDB	latin1_swedish_ci	80,0 KiB	0 B

Community

#	Kolonnenavn	Datatype	Kollation (Collation)	Attributter	Nulværdi	Standardværdi	Ekstra
1	id	int(3)			Nej	None	AUTO_INCREMENT
2	serial	varchar(31)	utf8_danish_ci		Nej	None	
3	password	varchar(31)	utf8_danish_ci		Nej	None	
4	name	varchar(63)	utf8_danish_ci		Nej	None	
5	bounds	varchar(127)	utf8_danish_ci		Nej	None	
6	users	varchar(255)	utf8_danish_ci		Nej	None	
7	create_date	datetime			Nej	None	

Home

#	Kolonnenavn	Datatype	Kollation (Collation)	Attributter	Nulværdi	Standardværdi	Ekstra
1	id	int(3)			Nej	None	AUTO_INCREMENT
2	address	varchar(511)	utf8_danish_ci		Nej	None	
3	coord	varchar(63)	utf8_danish_ci		Nej	None	
4	residents	varchar(255)	utf8_danish_ci		Nej	None	
5	type	varchar(15)	utf8_danish_ci		Nej	None	
6	embraceID	varchar(127)	utf8_danish_ci		Nej	None	
7	position	varchar(63)	utf8_danish_ci		Nej	None	

Event

#	Kolonnenavn	Datatype	Kollation (Collation)	Attributter	Nulværdi	Standardværdi	Ekstra
1	id	int(3)			Nej	None	AUTO_INCREMENT
2	community	varchar(7)	utf8_danish_ci		Nej	None	
3	title	varchar(255)	utf8_danish_ci		Nej	None	
4	description	varchar(4095)	utf8_danish_ci		Nej	None	
5	host	varchar(7)	utf8_danish_ci		Nej	None	
6	house	varchar(7)	utf8_danish_ci		Nej	None	
7	house_auth	tinyint(1)			Nej	0	
8	participants	varchar(255)	utf8_danish_ci		Nej	None	
9	start	datetime			Nej	None	
10	end	datetime			Nej	None	
11	create_date	datetime			Nej	None	
12	delete	tinyint(1)			Nej	0	
13	delete_date	datetime			Nej	None	

Item

#	Kolonnenavn	Datatype	Kollation (Collation)	Attributter	Nulværdi	Standardværdi	Ekstra
1	id	int(3)			Nej	None	AUTO_INCREMENT
2	community	varchar(7)	utf8_danish_ci		Nej	None	
3	value	varchar(15)	utf8_danish_ci		Nej	None	
4	unit	varchar(31)	utf8_danish_ci		Nej	None	
5	item	varchar(127)	utf8_danish_ci		Nej	None	
6	giver	varchar(7)	utf8_danish_ci		Nej	None	
7	takers	varchar(255)	utf8_danish_ci		Nej	None	
8	img	varchar(255)	utf8_danish_ci		Nej	None	
9	date	datetime			Nej	None	
10	create_date	datetime			Nej	None	
11	delete	tinyint(1)			Nej	0	
12	delete_date	datetime			Nej	None	

User

#	Kolonnenavn	Datatype	Kollation (Collation)	Attributter	Nulværdi	Standardværdi	Ekstra
1	**id**	int(3)			Nej	*None*	AUTO_INCREMENT
2	**name**	varchar(31)	utf8_danish_ci		Nej	*None*	
3	**mail**	varchar(63)	utf8_danish_ci		Nej	*None*	
4	**password**	varchar(31)	utf8_danish_ci		Nej	*None*	
5	**token**	varchar(63)	utf8_danish_ci		Nej	*None*	
6	**house**	varchar(7)	utf8_danish_ci		Nej	*None*	
7	**community**	varchar(255)	utf8_danish_ci		Nej	*None*	
8	**game**	tinyint(1)			Nej	1	
9	**status**	varchar(15)	utf8_danish_ci		Nej	mail	
10	**photo**	varchar(255)	utf8_danish_ci		Nej	images/User.png	
11	**count**	int(7)			Nej	*None*	
12	**last_active**	datetime			Nej	*None*	
13	**create_date**	datetime			Nej	*None*	