Architecture

2012-05-14

Contents

Versions

Version	Date	Changes	Performed by
1.7	2012-05-14	Spelling, formatting and fixed so it uses the english package	robud948
1.6	2012-05-13	Spelling and rephrasing.	nicol271
1.5	2012-05-13	Translated Document to english.	razli199
1.4	2012-05-02	Uppdaterat dokumentet så att det stämmer överens med arkitekturen.	oliuv486
1.3	2012-03-18	Fixat kommentarer från elaboration.	jonhe863
1.2	2012-02-21	Ändringar under elaboration.	jonhe863
1.1	2012-02-16	Gjorde om den enligt dokumentmall.	petla778
1.0	2012-02-11	Första versionen.	jonhe863

1 Purpose

This document describes the philosophy, decisions, restrictions, motivations and other aspects that shapes the design and implementation of the application. By continuously updating this document with information concerning both chosen implementation strategies, as well as those removed, no doubt shall exist concerning why a decision has been taken or for what reason.

2 Goal and philosophy of the architecture

By having conditions on a modular structure and documenting the code well, it will be easier to learn how the application works and continue its development. It shall be used as an example of how you can use the LiveDB API to build a peer-to-peer application. The web application shall work on cellphones, tablets and PC's with the only condition that they shall be able to use HTML5. The interface shall automatically adapt to the chosen platform which puts a lot of conditions on the application. The simplest way to handle this is to use a framework which supports adaption. The condition regarding the speed of the application is that it shall be responsive to the commands which the user can give.

3 Conditions and dependencies

Some members in the group has experience of web programming which should make the development phase a bit less problematic. Through their knowledge they can teach the rest of the group what they need to know in an effective way. The company Visiarc, to whom we shall develop the application, has developed a peer-to-peer based database API that we have gained access to. This means that we can focus on the design of the application instead of database development. However, the database is still in the development phase which means that we shall send feedback on the API during the development of our own application. Some functionality will be hard, if not impossible, to implement in our application if it is not supported in the API which we are supposed to build our application on.

4 Significant demands on the architecture

- R.1.2.0.1 - The response time for a page view shall be under 2 s.
- R.2.2.1.1 - Use Visiarc's LiveDB API.
- R.3.0.1.1 - Use HTML5, CSS and Javascript.
- R.3.0.1.2 - Use Node.js.

5 Decisions, limitations and motivations

1. We shall use LiveDB, the customers database, which is peer-to-peer based and handles the communication for the application.
2. We shall use Javascript. It is necessary since the customers framework is built in a way which requires this.
3. We have chosen to use Enyo2[1]. A group decision was made to use the Javascript framework Enyo2, since the customer mentioned it as a interesting technology to use. Enyo2 is a further development of Enyo which is a framework developed together with WebOS. It has recently been released as open source. The idea behind the framework is to create objects which can easily be re-used. Since the customer's database is using relatively new techniques we chose to write our application with the help of a new framework.
4. The customer wishes that each View (as in MVC) shall be in a separate file. Since the MVC pattern is hard to use together with the Enyo2 framwork, this pattern is not strictly used. The Enyo2 *kind* object (roughly Enyo's equivialent of an object oriented prototype) is the closest equivalent to an MVC view, which is why we have chosen to put each kind in a separate file.
5. We shall not use global variables and use a clear namespace for CSS and id in generated HTML. This is to create the possibility to run multiple applications which uses LiveDB on the same webpage.

[1] `http://www.enyojs.com`

6 Architecture principles

1. **Parallellism** - It shall be possible to run multiple LiveDB applications on the same webpage.

   By avoiding using global variables and naming namespaces for CSS and generated HTML with a clear prefix several applications can be run on the same webpage.

   Status: Implementation remains.

2. **Localization** - Support for both English and Swedish shall be implemented.

   Putting all the text strings as variables in a .js-file and adding the file to the package.js will make it possible to use the strings from files that contains the implementation of the framework. It is important that the strings in the document are grouped by view, so it is easy to find the strings for a specific view.

   Status: Implementation has begun.

3. **Debugging** - Support for testing should be implemented

   With a close cooperation between the testleader and the rest of the group the application shall be developed in such way that it promotes testing.

   To make the unit testing easier the codes shall be written modular with several functions that does little stuff instead of a big function that does everything.

   Status: It is possible to run automatic tests for both server and client.

4. **Error handling** - Some error handling for indata from users.

   The customer has the responsibility to handle bad indata to the database which could destroy it. Simpler error handling like empty strings will be handled by the application.

   Status: Implementation remains.

5. **Design** - The code shall be well documented and structured in a professional way. See the chapter on Design.

   Status: Done.

6. **Security** - Even though we will not develop the application so that it will support multiple security levels, it shall still be easy to implement

when the functionality is added. It shall for example be possible to implement moderator-status etc.

To implement this it is required that the different layers are stored in the database. Unfortunately, in the current situation this could be manipulated locally. It means that users can practically change their security level to whatever they want. This is not something that we affect but requires the database to evolve.

We shall write our application so that it will be able to handle different security levels. This we will do by depending on different attributes different views will be loaded depending on the users security level. The probability is high that only functions for a users-level will be implemented when the project has ended.

Status: Implementation remains.

7. **RESTfulness** - Each page shall have a unique adress; in that way standard browser functions like backwards/forward and save bookmark will work as expected.

   Status: Implementation has begun.

8. **Platform independent** - The application shall work on cellphones, tablets and PC's in web browsers that supports HTML5.

   We will firstly create a working application on cellphones and afterward, depending on how much time we have available, continue working on the other platforms. We have chosen to use a framework that shall be able to scale well. If it does not work in a satisfying way in all situations it can be solved by different CSS-files depending on which platform are to be used.

   A test page scales in a satisfying way when a android cellphone visited the page when we used a prototype.

   Status: Implementation remains.

7 Keywords for the application

- Categories: The forum is split into different categories. This means that all threads are not put in one place which creates a good overview and structure over what kind of discussion topics that are desired.
- Threads: A specific subject is discussed within a thread, the subject is specified in the title of the thread.

- Posts: A thread consists generally by a number of posts that comprises a thread start with answers and and discussions regarding the subject.
- Users: To be able to write a post it is required that you have created a user on the forum.

8 Design

We are using the framework Enyo.js and our code reflects this. Enyo uses prototype based object orienting, where prototypes are called kinds. Every kind handles its own in- and output and has through Enyo an automatic connection to the DOM. Following things are characterizes the design:

server/server.js

The node.js server we use. Specifies rules for how we answer HTTP requests.

forum/index.html

The page which is serviced by the node server. Includes the client side resources our application consists of.

package.js

This file points out the files in each map that shall be used by the application and in which order they shall be included.

forum/forumdatabase.js

A service which handles all access to the distributed database. Can be called by any other code in the application. Calls are done asynchronously by a callback function, which also continue running as soon as relevant data is updated. This results in push based updating that makes it possible for us to automatically react to new data.

forum/kinds

Here are the Enyo kinds that is the application. In the top of the hierarchy is Forum.App.js. It uses all other kinds. A kind defines (with prototype) an object, which automatically can represent itself in the DOM and include other kinds within itself.

9 Architectural views

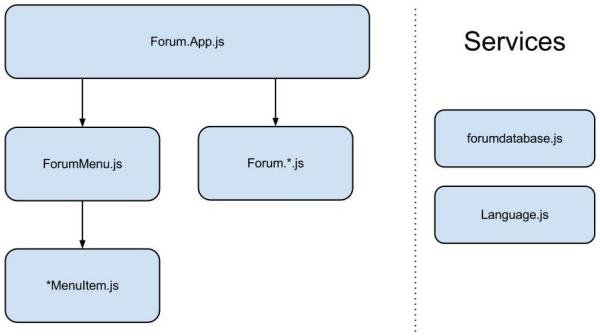

Figure 1: Module overview, client side

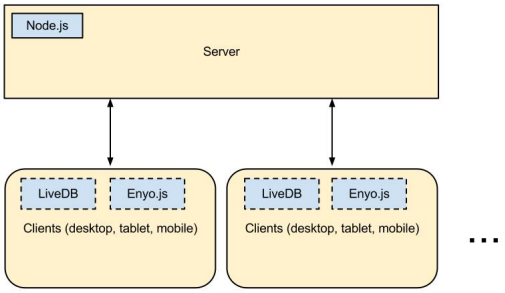

Figure 2: Deployment view, data flow diagram

10 Rejected design ideas

10.1 Model View Controller

Before we started using Enyo we thought that it would be smooth and easy to use the design pattern Model-View-Controller(MVC), since it is often used in popular web frameworks (e.g Django). We started the project by following guides and similar resources to our framework, Enyo. It resulted in that an explicit design pattern, which fitted the framework better, was used instead of MVC.

Enyo makes the programming a lot easier even though we don't use MVC. Enyo solves most of the work with the DOM. This is most of what shall be handled of a MVC view. A part of the information about the DOM remains in the model (which mostly consists of Enyo kinds), code for presentation and interactivity is strongly connected which makes it hard to separate Model and Controller.

We still have many conditions to fulfill, and to modify our code so that it follows a MVC pattern would not make the rest of the work any easier. More likely, it would make the purpose of the code harder to comprehend. We have therefore chosen to remove the condition that our application shall be written accordingly to MVC.