BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program BEW

Development of modern Web interfaces for embedded systems

By: Alexander Peters
Student Number: 1310255025

Supervisor 1: Dipl. Ing. Christoph Veigl

Vienna, 2015-6-2

FACHHOCHSCHULE TECHNIKUM WIEN

Declaration of Authenticity

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (for example see §§ 21, 46 and 57 UrhG (Austrian copyright law) as amended as well as § 11 of the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

In particular I declare that I have made use of third-party content correctly, regardless what form it may have, and I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see § 11 para. 1 Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.”

Place, Date	Signature

Kurzfassung

Obwohl sich die Möglichkeiten in den letzten Jahren stark weiterentwickelt haben, werden diese im Bereich der eingebetteten Systeme kaum genutzt.
Diese Arbeit gibt einen Überblick über die neuen Technologien und die möglichen Anwendungen. Weiterst werden diese den traditionellen und weiterhin genutzten statischen Webseiten gegenübergestellt.

Um die Möglichkeiten und den nötigen Aufwand zu evaluieren, wird ein kompletter Webservice-Stack mit einer modernen single-page Webapplikation entwickelt. Als Basis dieser Entwicklung dient smarte Tischfußballtisch der Mensa der Fachhochschule Technikum Wien. Die dabei auftretenden Problem werden erörtert und mögliche Lösungen aufgezeigt.
Zusammenfassend kann gesagt werden dass moderne eingebettete Systeme ausreichend Ressourcen haben um von den neuen Technologien zu profitieren. Auch sind entsprechende Endgeräte zur Nutzung des Interfaces auf breiter Basis verfügbar.
Es steht eine Reihe guter Frameworks zur Verfügung um die Entwicklung zu vereinfachen und zu beschleunigen jedoch ist selbst unter Einsatz dieser Frameworks die Entwicklung immer noch sehr aufwändig und die benötigten Fähigkeiten weichen stark von denen eines embedded Developers ab. Daher ist es ratsam eine Person im Team speziell für Webentwicklung zu haben oder diese Aufgaben an ein spezialisiertes Team abzugeben.

Schlagwörter: Web, Interface, HTML, embedded systems, Webserver, Node.Js

Abstract

While the possibilities of web applications have evolved greatly in the last years, most embedded systems still don’t make use of those new opportunities.
This thesis gives an overview of the new technologies and their possible applications and compares them to the traditional and still widely used static pages.

To evaluate the possibilities and the effort needed to integrate those technologies a complete web service stack with a modern application is implemented on top of an existing smart tabletop soccer equipped with an raspberry pi. The problems and challenges are discussed and possible solutions are shown.

In conclusion it can be said that modern embedded systems do provide the resources to make use of those technologies and compatible end devices to consume the interface are widely deployed. Also there is a wide range of usable open-source frameworks available, which can be implemented, in commercial projects. However the complexity of rich single-page web applications, even when using those frameworks is very high and the necessary skills are very different from the skillset of a traditional embedded developer. Therefore it is advisable to have a dedicated web developer or outsource those tasks to a specialized team.

Keywords: Web, Interface, HTML, embedded systems, Webserver, Node.Js

Table of Contents

1 Introduction

Embedded systems are getting more and more widespread. While some devices like Smartphones and Tablets have there own screens, most embedded systems do not have their own user interface-hardware. The wide availability of internet-enabled devices makes web interfaces a viable choice for embedded devices user interfaces.

While the possibilities offered by browsers have greatly evolved in the last years, most embedded devices like routers, are still stuck with static web pages and server-side application logic.

This work gives an introduction into the development of modern singe-page applications and their benefit for embedded device interfaces.

The smart tabletop-soccer in the canteen of the UAS Technikum will serve as test bed and demonstration object.

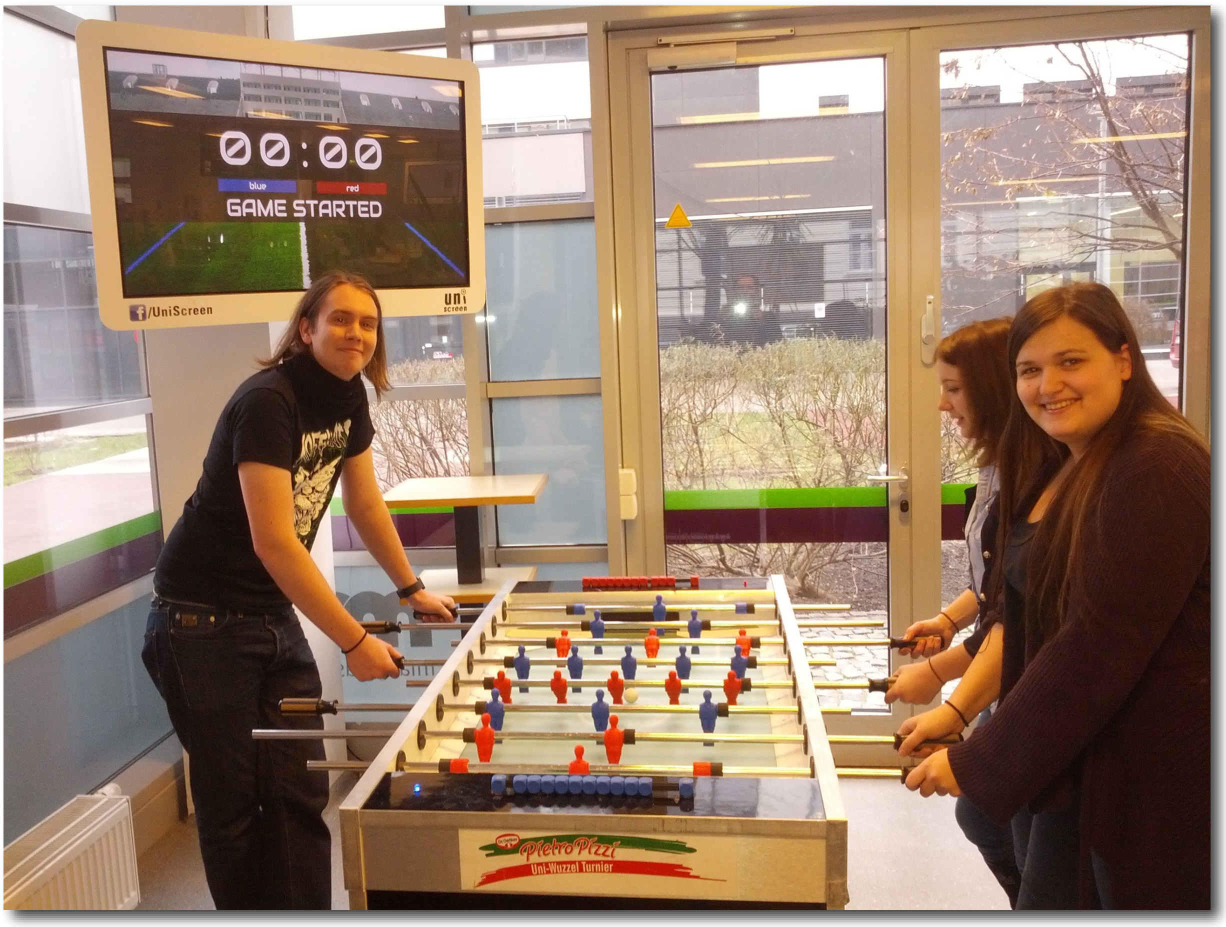


Figure 1: tabletopsoccer, canteen UAS Technikum Wien

1.1 Preconditions

The tabletop soccer, located in the canteen of the UAS Technikum Vienna, was equipped with light barriers, buttons and a Raspberry Pi in a previous project. The detailed state of the system as of beginning of this project is documented in the corresponding bachelor thesis.

The most important facts are:

- 2 light barriers to detect goals
- 2 buttons for user interaction
- 2 software controllable LEDs (integrated with the buttons)
- Arduino Mega
- Raspberry Pi Model B
- HDMI output with attached FullHD Screen
- Wi-Fi access point

1.2 Feature requirements

In addition to the existing features the following functionalities have to be implemented.

1.2.1 User Management

Using the smartphone client it hast o be possible to register as a new user with a username and password or to login as existing user.
When returning to the page the session should be restored to avoid reentering username/password.
If desired the user hast to be able to log out.

1.2.2 Challenge players

A user must be able to see the other players currently online and must be able to challenge them. When a user receives a challenge and accepts it a new game is either started or added to the waiting list.

1.2.3 Game Records

After a non-anonymous game is finished, the results have to be stored in the database.

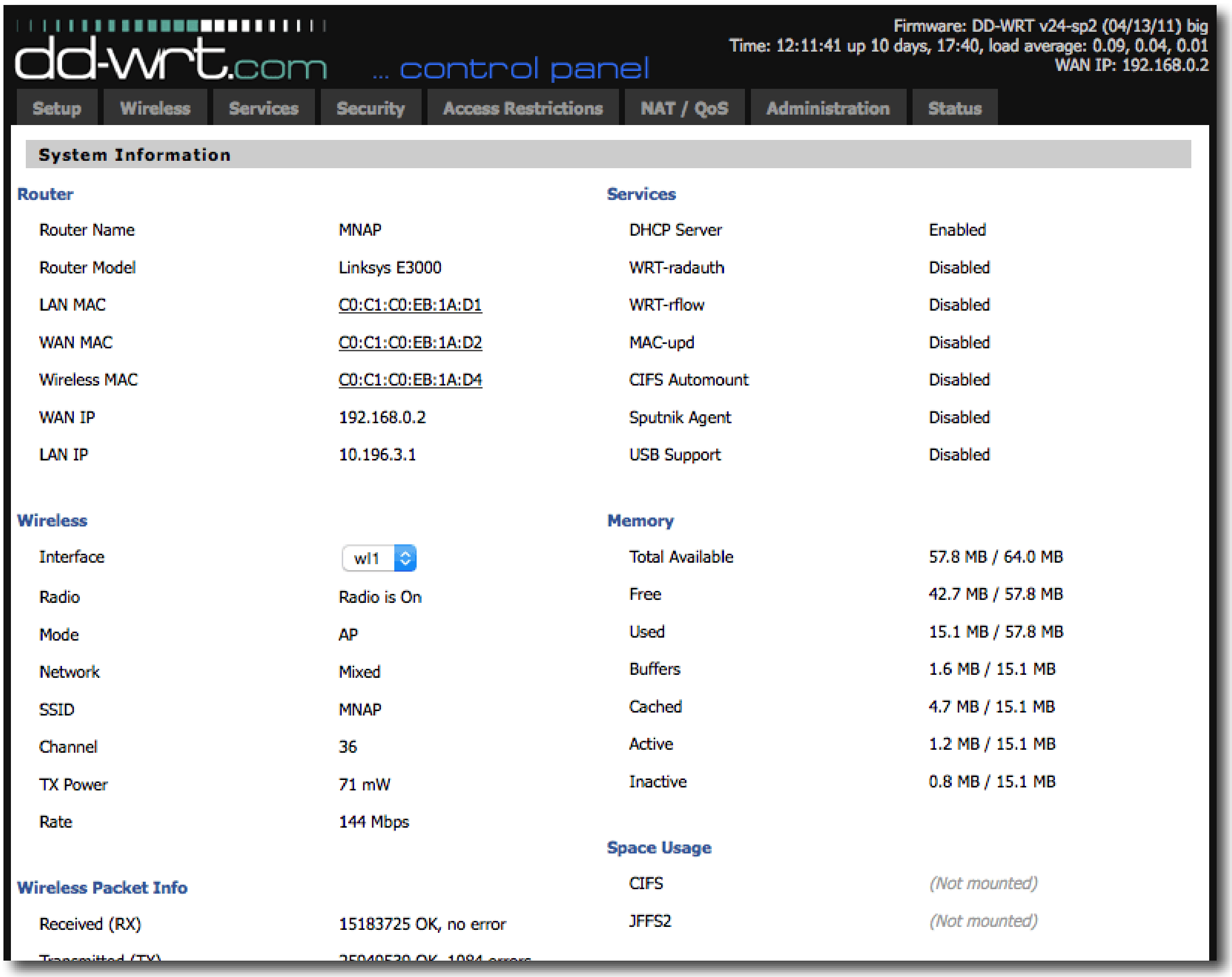
When no game is in progress the Top10 players of the current day, week, month, year and since beginning of record should be displayed alternatingly.
In addition every user should be able to see his statistics using the client interface.

2 Static Pages vs. Applications

Since the beginning of the Internet and the appearance of the first browsers in the 1990s the possibilities for web pages have greatly evolved. Being purely static pages in the beginning the modern web applications can now compete to native rich client applications.
While this development has been accepted and is now widely used by most media companies and web developers and many embedded appliances provide a web interface for configuration and maintenance, most of those interfaces are still stuck on an evolutionary step from about the year 2000

2.1 Pure static content

In the beginning of the Internet webpages were static. They were written in HTML, stored on a server and provided under a unique path. When a client requested this Unique Resource Location (URL) or Unique Resource Identifier (URI) the corresponding file was read from the Servers hard drive and delivered to the client without changes. The different pages were connected via static links.



Firmware: DD-WRT v24-sp2 (04/13/11) big
Time: 12:11:41 up 10 days, 17:40, load average: 0.09, 0.04, 0.01
WAN IP: 192.168.0.2

System Information

Router

Router Name	MNAP
Router Model	Linksys E3000
LAN MAC	C0:C1:C0:EB:1A:D1
WAN MAC	C0:C1:C0:EB:1A:D2
Wireless MAC	C0:C1:C0:EB:1A:D4
WAN IP	192.168.0.2
LAN IP	10.196.3.1

Wireless

Interface	wl1
Radio	Radio is On
Mode	AP
Network	Mixed
SSID	MNAP
Channel	36
TX Power	71 mW
Rate	144 Mbps

Wireless Packet Info

Received (RX)	15183725 OK, no error

Services

DHCP Server	Enabled
WRT-radauth	Disabled
WRT-rflow	Disabled
MAC-upd	Disabled
CIFS Automount	Disabled
Sputnik Agent	Disabled
USB Support	Disabled

Memory

Total Available	57.8 MB / 64.0 MB
Free	42.7 MB / 57.8 MB
Used	15.1 MB / 57.8 MB
Buffers	1.6 MB / 15.1 MB
Cached	4.7 MB / 15.1 MB
Active	1.2 MB / 15.1 MB
Inactive	0.8 MB / 15.1 MB

Space Usage

CIFS	(Not mounted)
JFFS2	(Not mounted)

Figure 2: traditional web interface on a Linksys router

2.2 Server generated content

The next step was to introduce server side logic to process information submitted via the request (for example from a form) and other information like server side databases and build the page delivered to the user dynamically. Now the server holds just templates for the different pages and when a page is requested it generates the actual content to be delivered based on the template and the specific data sources. The most common technology for server side logic used to be and still is PHP.
Even tough now pages could integrate user specific or otherwise dynamic content they are still immutable once they are delivered. The only way to change the view in the browser was to do another request to load an updated version of the page.

Most web interfaces of systems like routers are still stuck on this level of evolution.

2.3 Dynamic pages

With the introduction of client side scripting language it became possible to deliver a page, which can change dynamically due to user action. This means that it is not longer needed to make a lot of server requests to change the appearance of the active site. This enables things like hiding or animating menus, client side validation of forms or other user-supplied data or even the most annoying ads ever seen.
In the beginning of those dynamic pages there was a lot of different technologies like VBScript by Microsoft, Macromedia (now Adobe) Shockwave and Flash and Netscape's LiveScript. While the former ones were proprietary and therefore causing a lot of compatibility problems, LiveScript became an open standard and became the now still used JavaScript or EICMA Script language.
With the advent of HTML5 and the most recent JavaScript implementations there is basically not need for any proprietary add-on and many of them have been discontinued finally.

2.4 Single page web application

While using basically the same technologies as dynamic pages a single page web application differs by loading the whole application at the beginning and then only fetches data from the server to render it on the client. While this increases the initial load time it leads to a way more responsive user experience since switching between views does not require an additional request to get the next page.
Single page web applications are quite similar to traditional rich client applications except that they don't have to be installed first. Using the extended caching options introduced along with HTML5 even the initial load time can be eliminated in most circumstances.

3 Web Frameworks

Modern Web applications can be very close to native applications. This has been mode possible by new standards like HTML5 and CSS3 and a lot of features available in the browser. While this provides great options to develop applications it also adds a lot of complexity to a project.
To be able to make use of this rich set of features while keeping the development time affordable there have been a variety of frameworks developed.
While development of complex applications gets simplified, the use of a framework initially adds considerable overhead. To cope with that the use of known to work boilerplates or app generators like yeoman is recommended.

This chapter will give an overview about the most widely used frameworks.

3.1 AngularJS

AngularJS[1] is a client-side framework developed by Google in 2009.
The general aim of the project was to close the gap between HTML, which was initially developed for static pages, and native applications. It takes HTML as a template-language and extends it by so-called directives to allow for dynamic and programmatically interaction with the DOM.
It also provides controllers to define the behavior of the as well as Services to interact with external resources like REST interfaces or other types of APIs
For a more detailed guide please refer to the official documentation [2]

3.2 ReactJS

ReactJS is a performance oriented view-rendering library written in JavaScript. It was originally developed by Facebook and first released in 2013. Other than Angular it does not care about the model or controller from the MVC pattern but leaves those tasks to other projects. It can also be used together with AngularJS.

The specialty of React is the virtual DOM. Because DOM access and rendering are very slow React keeps its own copy of the DOM for interactions and generation of DOM patches to apply to the real DOM.

3.3 Foundation

Foundation [3] is a Framework for creating responsive mobile-first webpages and applications. It is not comparable to the previous discussed frameworks but provides UI elements to use with any MVC framework including AngularJS.
As it is a responsive framework it allows using the same UI on different screen sizes with good user experience.
An overview of the available components can be found here [4]

For use with AngularJS there are special AngularJS directives available from Pinecone [5]

3.4 Bootstrap

Bootstrap [6], originally named Twitter Blueprint, was developed by Twitter and first released in 2011. From there it has greatly evolved and now, just like Foundation, Bootstrap is a Framework for building responsive, mobile first Web applications.
Like Foundation it provides a large collection of predefined components [7]

4 Server <-> Client communication

Having traditional web pages the only communication between server and client was the request to get a new page, which was then answered either by sending the desired resource or by an error code. The only possible user interaction was to follow a link, which will request the next page containing other links.

With the introduction of client side logic it became necessary to load additional content after a page is delivered. For instance when filling a sign up form the website can check if the username is available without submitting the form using a server side API call.

While requesting additional data from the server is relatively simple it has been a problem to push data from the server to the client for a long time.

4.1 Technologies

This chapter gives an introduction about the different technologies for communication between a server and a script running in the browser.

4.1.1 XML HTTP Request (XHR)

XML HTTP Request (XHR) was introduced by Microsoft but was quickly adapted by all other major browsers. The name is misleading since the requests do not need to contain XML but will work for every kind of data. XHR allows Scripts running in the browser to make arbitrary HTTP request to a server. This technology builds the foundation for AJAX (Asynchronous JavaScript and XML).

4.1.2 XHR Polling and Long Polling

While XHR allows to send data from the client to the server or to request additional data from the server it does not allow to push data from the server to the client in real time. When a client waits for an event on the server it has to poll for it. This means it periodically request a resource and receives an empty response if there is no new data on the server.
This means that there may be a significant lag between the occurrence of the event and the client being aware of the event. XHR Long Polling works almost the same but tells the server not to deliver an empty response but keeps the connection open until a new event occurs. Then the connection is reestablished and waits for the next event (see Figure 3).
This reduces the number of connections to establish and allows for near real time event signaling. IBM published a deeper discussion on this topic [8].

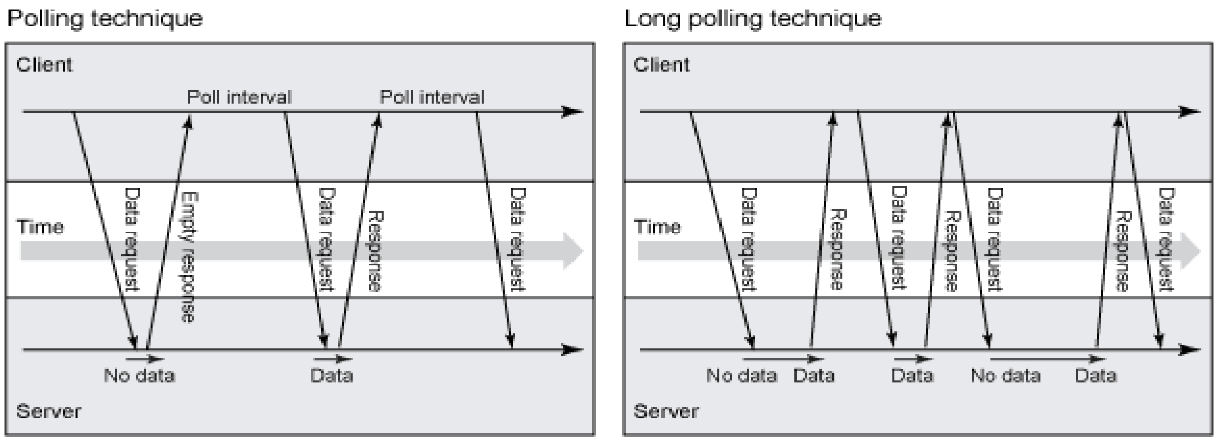


Figure 3: Polling vs. Longpolling [8]

4.1.3 Websockets

Websockets [9] are a relatively new addition to the HTTP standard. Once enable it enables bidirectional communication. To establish a websocket connection the client sends an HTTP request and instructs the server to upgrade this connection to a websocket connection. If the server supports this, it will confirm the upgrade and the connection stays open for messages from both parties.

4.2 Libraries

To simplify the use of previously mentioned technologies a couple of libraries have been developed. All of the mentioned libraries consist of a node module for the server and a client module for use in all major browsers.

4.2.1 Socket.io

Socket.io [11] was the first big player to allow for simple implementation of a websocket connection. It also features a lot of fallback possibilities back to XHR polling for situations where websockets are not available.

4.2.2 Primus

Primus [12] is a Framework to utilize different transport mechanisms including socket.io. The transport used can be selected at initialization. This allows changing the transport without touching the overlaying code.

4.2.3 Eureca.io

Eureca.io [13] is built on to of Primus. It allows sending remote procedure calls (RPC) over the connection supplied by primus and takes care of request/response mapping. This project is quite new and not as well maintained as Primus and socket.io.

5 Persisting Data

In most systems data has to be persisted to survive a reboot of the device. The amount and the structure of the data, the necessary level of consistency and integrity, the way it is accessed as well as the systems resources have to be considered when choosing in which way those data will be stored.

5.1 File system

Writing data into a file is the easiest way of persisting data. It is does not require any additional service running on the system and is therefore very lightweight. However this introduces complexity like file parsing to the application. Also it does only work well for either small amounts of data or data that is not accessed randomly.
If data has to be accessed randomly or has to be processed and searched a database system should be used.

5.2 Relational Database Management Systems (RDBMS)

One of the most common type of database are the relational database management systems (RDBMS). An RDBMS stores normalized data in 2 dimensional tables that can be filtered and joined together on demand. To define, add and query data from a relational database the structured query language (SQL) is used.

Most RDBMS are ACID-compliant. That means the guarantee Atomic, Consistent, Isolated and Durable operations. Those properties are very important when multiple users access a common set of data concurrently.

The most prominent implementations are the proprietary Oracle database as well as the open-source projects PostgreSQL[14], MySQL[15] and its fork MariaDB[16].
While MySQL and Maria DB are widely deployed as part of the LAMP stack they lack proper ACID compliance and big parts of the SQL standard like, recursive queries, are not implemented. PostgreSQL offers full ACID compliance, very close adherence to the SQL standard and a lot of enterprise ready replication options.
PostgreSQL a well as MySQL can be configured to run even on low-end embedded Linux systems.

5.3 NoSQL Databases

NoSQL Databases are a very wide field. It includes every Database that does not or not only support SQL. Usually those systems are also not strictly ACID compliant.
Two of the most common noSQL databases are described in this chapter.

5.3.1 Key-Value Store

A key-value store is the simplest of all data structures. It just needs to support 2 operations: writing a value for a specified key and lookup the value for a specified key. Key-value stores are usually implemented as hash tables. That makes access very fast and allows for easy distribution across multiple machines.
A widely used open-source implementation is the Redis[17] key-value store.

5.3.2 Document Store (MongoDB)

MongoDB[18] is document-oriented database with optional dynamic schema validation. That means other then relational databases it does not store datasets as lines in tables but it stores JSON documents (objects). These may be optionally checked by a scheme validator.

Since MongoDB uses JSON it works very well with JavaScript and therefore with web applications.

6 Create a Web application

This chapter gives a basic introduction on how to create a web application using AngularJS and Foundation. As build tool gulp will be used.
Please note that all the steps and commands in this chapter are executed on the development machine and not on the target system. While it is possible to install the tool chain right on the target as well, this is not recommended since some parts are quite resource intense.

6.1 node.js

Since this tool chain is based on node.js it has to be installed first. Since the way to install node.js is dependent on the used platform there are no exact steps given. Detailed instructions can be found on the node.js website [10].

6.2 yeoman gulp-angular

Building a web application from scratch is a quite complex repetitive task. This issue is addressed by scaffolding tools like yeoman[19]. Yeoman is a framework for code generators. This project will be build using the gulp-angular generator [20].

First yo, gulp [21] and bower [22] need to be installed. Then the generator can be installed.
Since those are not specific to the project but general utilities that may be used across projects they are installed globally using the –g flag.

```
#> npm install -g yo gulp bower
#> npm install -g generator-gulp-angular
```

6.3 Preparing the directory

Once the tool chain is installed the generator can be used to create the basic application structure. Therefore a new directory is created:

```
#> mkdir wuzzler_client
#> cd $_
```

It is very advisable to put the new directory under version control. Git is a widely used and currently the most advanced versioning tool. To create a new repository just call:

```
#> git init
```

to exclude platform dependent binaries and build output from the repository there should be a **.gitignore** file added to the repository:

```
node_modules/
bower_components/
.sass-cache/
.tmp/
dist/
```

6.4 Scaffolding the application

In the new directory the generator can be called:

```
#> yo gulp-angular
```

It will then prompt for some options on how to create the application. For this project we will use the following options:

? Which version of Angular do you want?
1.3.x (latest)
? Which Angular's modules would you want to have?(ngRoute and ngResource will be addressed after)

- *angular-animate.js (enable animation features),*
- *angular-cookies.js (handle cookie management)*
- *angular-touch.js (for mobile development)*
- *angular-sanitize.js (to securely parse and manipulate HTML)*

? Would you need jQuery or perhaps Zepto?
None (Angular will use its own jqLite)
? Would you like to use a REST resource library?
None, $http is enough!
? Would you like to use a router ?
UI Router, flexible routing with nested views
? Which UI framework do you want?
Foundation, "The most advanced responsive front-end framework in the world"
? How do you want to implements your Foundation components?
Angular Foundation, a set of native AngularJS directives based on Foundation's markup and CSS
? Which CSS preprocessor do you want?
Sass (Node), Node.js binding to libsass, the C version of the popular stylesheet preprocessor, Sass.
? Which JS preprocessor do you want?
None, I like to code in standard JavaScript.
? Which html template engine would you want?
None, I like to code in standard HTML.

Once the generator has finished the directory contains a working application (see Figure 4) to start from.

Before starting any changes the current state should be committed to the git-repository to allow stepping back if needed:

```
#> git add -all
#> git commit -m “add basic application structure“
```

6.5 Live preview

To see what the application looks like call the command:

```
#> gulp serve
```

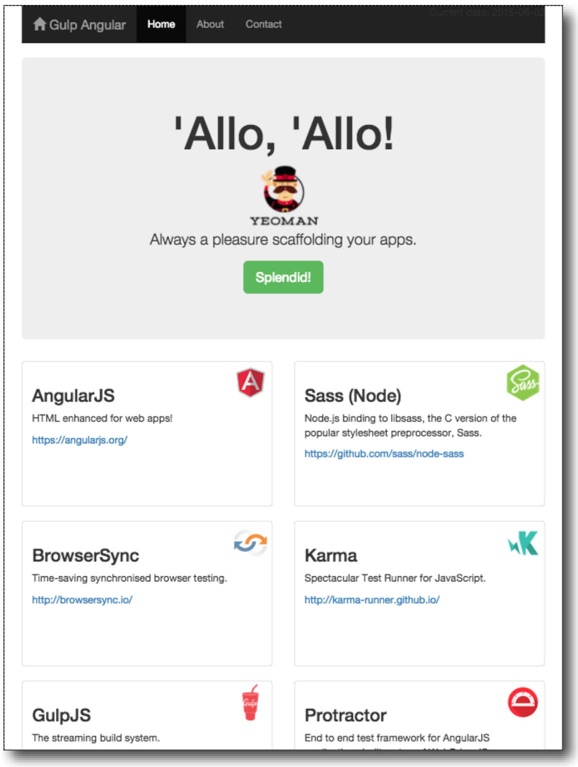
This command will compile all the files, start a webserver and open the application in the default browser. Also it will constantly monitor the files served and update reload the browser on every change. This works reliably for changes in files but often misses newly added files. So after adding or deleting files gulp should be stopped using ctrl+C and started again

6.6 Build application

To create a distributable version of the application, run the command:

```
#> gulp
```

This command will compile and minify all the files and puts the output into the **./dist** folder. This directory can then be copied to the public directory of the webserver



'Allo, 'Allo!

YEOMAN

Always a pleasure scaffolding your apps.

AngularJS
HTML enhanced for web apps!
https://angularjs.org/

Sass (Node)
Node.js binding to libsass, the C version of the popular stylesheet preprocessor, Sass.
https://github.com/sass/node-sass

BrowserSync
Time-saving synchronised browser testing.
http://browsersync.io/

Karma
Spectacular Test Runner for JavaScript.
http://karma-runner.github.io/

GulpJS
The streaming build system.

Protractor
End to end test framework for AngularJS

Figure 4: gulp-angular template

7 Implementation

This chapter gives an overview about the implementation of a web service stack based on the tabletop soccer in the canteen of the UAS Technikum.

The project is separated into 3 components that will communicate over the local network. Each component has an own git repository and its own dependencies and build chain. The components are described in the following sub-chapters.

7.1 Backend

The backend is built on top of Node. It communicates with an attached Arduino, controlling the light barriers, using cylon.js. Node.js was chosen because its simplicity and wide community. Also being able to code in JavaScript on the backend makes it easy to use the same code on frontend and backend side. Furthermore Node.js has the probably best support for real-time communication using websockets.
As communication Library Eureca.io is used because it allowed for simple RPC over websockets.

The data is stored in a PostgreSQL 9.1 database. Although Node.js is mostly used in conjunction with MongoDB, a relational database was chosen because of the better support of analytic functions. The data design can be seen in Figure 5.

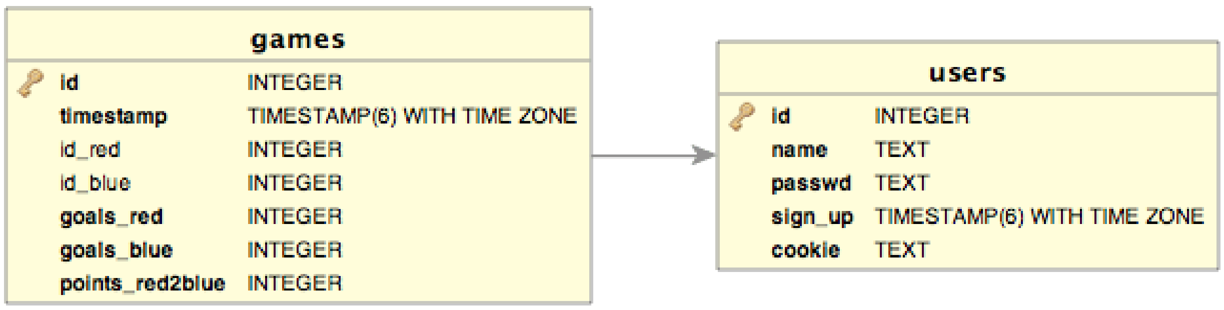


Figure 5: E/R model

7.2 Display

The display web interface is meant to be displayed on one or more screens without user interaction.
While a game is in progress it will show the current state of play(see Figure 6) . When there is no game running it will display the Top10 Players of the current day, week, month, year and since begin of the records (see Figure 7).
The application is built using the gulp-angular generator described in chapter 6.

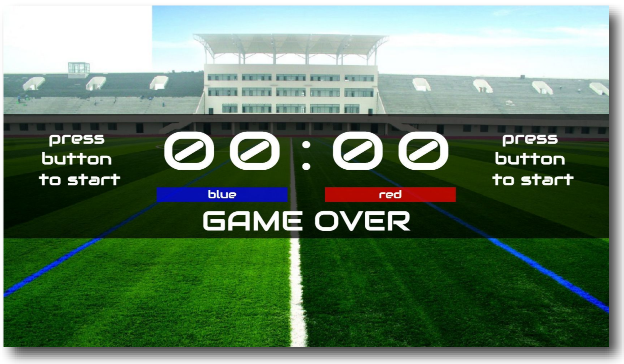
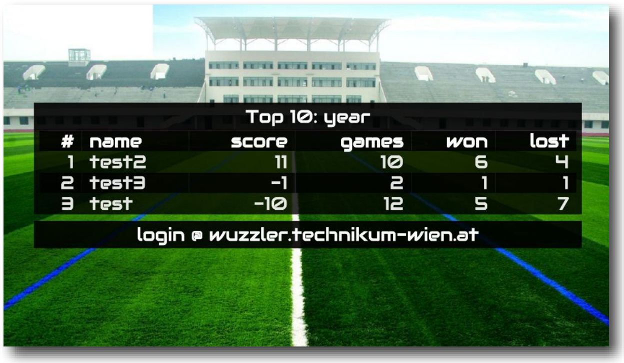


Figure 6: in game display

The 2 modes, game and statistics view are implemented as independent views within the same app. Routing is accomplished using the ui-router module.

The application connects to the backend using eureca.io. In gaming mode it waits for updates to be pushed and will then refresh the view.

In statistic mode it will actively query the necessary Data from the server.


Top 10: year

#	name	score	games	won	lost
1	test2	11	10	6	4
2	test3	-1	2	1	1
3	test	-10	12	5	7

login @ wuzzler.technikum-wien.at

Figure 7: statistics display

The layout - including font size – is dependent on the width of the device rendering the application. So the application looks the same regardless of the actual resolution. This was tested for 1280x720 as well as 1920x1080. It will also work for other resolutions as long as the height to with ratio is not wider than 16:9. Also the browser needs to support the unit “vw”.

7.3 Client

The client Interface is meant to be used on the users smartphone to login, start games and view own statistics. Although the interface can be used on desktop browsers as well it is mainly targeted to smartphones.
As the display interface it is built using the gulp-angular generator described in chapter 6.
Big parts of the backend communications are identical to the code used in the display module. However some calls were added to allow for the necessary user interaction.

On load the application checks for a valid session token in the local browser storage. If a token is found it will then send it to the server to restore a previously opened session and jump to the profile view. If no token is found, or the token is rejected by the server, the login screen (see Figure 11) will be displayed.

Just as the display module it will receive updates from the backend. Those pushed updates are used in the start game menu to update the list of challenges and available users (see Figure 10).
The statistics in the profile view (see Figure 9) are actively queried from the server every time the menu is opened.

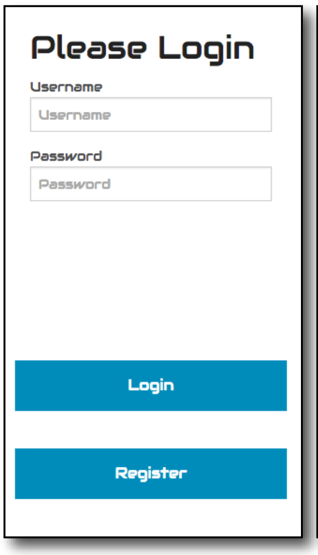
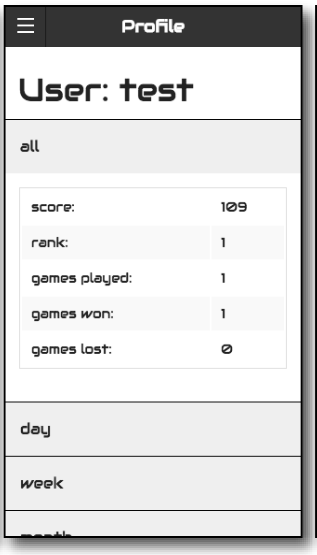


Figure 11: login screen



score:	109
rank:	1
games played:	1
games won:	1
games lost:	0

Figure 9: profile view

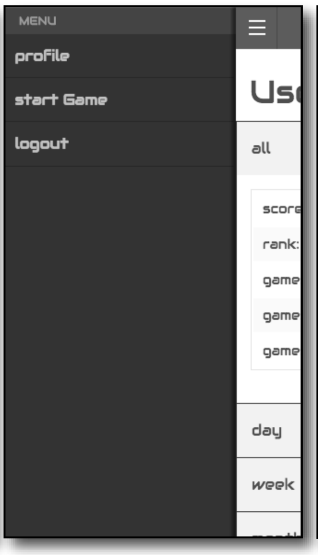
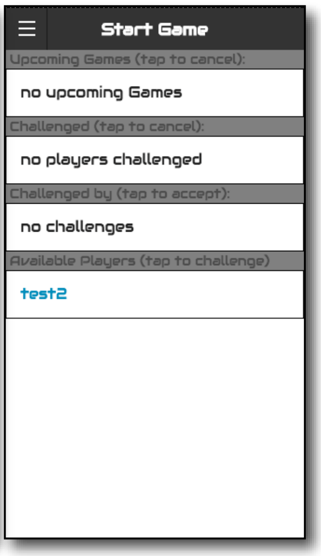


Figure 8: off-canvas navigation



Upcoming Games (tap to cancel):
no upcoming Games
Challenged (tap to cancel):
no players challenged
Challenged by (tap to accept):
no challenges
Available Players (tap to challenge)
test2

Figure 10: start game menu

8 Setup and Deployment

8.1 Copy files

This project consists of 3 repositories. The **wuzzler_backend** repository contains all backend components including the node.js program, the script to create the database and the autostart scripts. It has to be copied to the home directory of the user pi: **/home/pi/wuzzler_backend**

In addition to the backend there are 2 frontend repositories. The **wuzzler_display** holds the application, which will be shown on the screen and **wuzzler_client**, is the user interface supposed to be used on the player's smartphones.
Those 2 applications have to be built and the resulting **dist** folders are to be placed under **/home/pi/wuzzler_backend/public/display** and **/home/pi/wuzzler_backend/public/client**

8.2 NTP

Since the Raspberry Pi does not have a hardware clock the time must be set on every boot. This can be done via the network time protocol (NTP) [23]. NTPd is a daemon that runs in the background and constantly syncs the system clock against a list of timeservers.
NTPd can be easily installed via aptitude:

```
#> aptitude update
#> aptitude install ntpd
```

8.3 Node.js

Node.js is already preinstalled but as this project introduces a lot of new dependencies it needs to be upgraded avoid compatibility problems later on.
As the packages in the official repository are even older than what is deployed now, the package manager is not much of help in this case. Fortunately there are at least binaries available[24] so compiling from source can be avoided.

These download and extract the latest version of node:

```
#> wget http://nodejs.org/dist/v0.10.28/node-v0.10.28-linux-
arm-pi.tar.gz
#>tar -xzf node-v0.10.28-linux-arm-pi.tar.gz
```

Before moving the binaries to their final location the old binaries are backed up:

```
#> mv /opt/node /opt/node.bak
#> mv node-v0.10.28-linux-arm-pi /opt/node
```

Before the application can be started the dependencies need to be installed:

```
#> npm install
```

Port forwarding and autostart have already been implemented before – only the path of the app has to be updated. Therefore the corresponding line in the file ***/etc/rc.local*** has to be changed:

```
#start wuzzler app on boot
su pi -c "/opt/node/bin/node
/home/pi/wuzzler_backend/app.js" &> /var/log/wuzzler.log &
```

8.4 PostgreSQL

PostgreSQL is available from the official repository in version 9.1 while the current version is 9.4. As there are no precompiled binaries for a more recent version than 9.1 the only possibility is to compile from source.
While PostgreSQL introduced some really handy features like JSON handling and named parameters for functions, compiling it from source on a raspberry pi takes a lot more time and is likely to introduce more problems than to work around those missing features for this relatively simple application.

Therefore the packages are installed using aptitude. Note that the package **postgresql-contrib** is needed as well to provide UUID support.

```
#> aptitude update
#> aptitude install postgresql postgresql-contrib
```

After installing postgres the user and the database have to be created. This has to be done as the user “postgres”

```
#> sudo su postgres
#> createuser pi
#> createdb wuzzler_db
#> exit
```

Now the setup script can be run to create all tables and functions needed by the application:

```
#> psql wuzzler_db -f wuzzler_backend/wuzzler_db_9.1.sql
```

8.5 Chromium Autostart

Because oft he additional dependencies node.js now takes more time to start. Keeping the old autostart mechanism this leads to the browser being started before the server is ready, which will in turn only display a server not found error.

To cope with that the browser is not started directly using the lxde autostart configuration but instead a terminal is launched to execute a script, which waits for the server to become available and then starts the browser.
Therefore in the file ***etc/xdg/lxsession/LXDE/autostart*** the line starting chromium has to be replaced like this:

```
@lxterminal -e “/home/pi/wuzzler_backend/start_chrome.sh“
```

8.6 Migrating to Raspberry Pi 2

While the backend runs just fine on the Raspberry Pi (Figure 13), Chromium causes up to 100% CPU Load and still needs up to 30seconds to render the Webpage.

Therefore the installation will be migrated to the upgraded version of the Raspberry Pi. It features a faster processor and more RAM (see Figure 12).

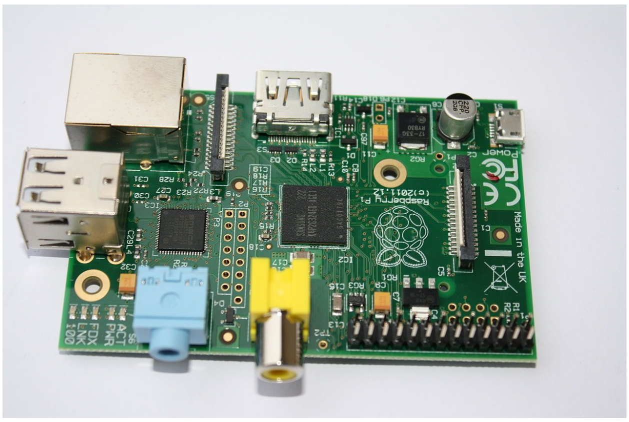


Figure 13: Raspberry Pi B

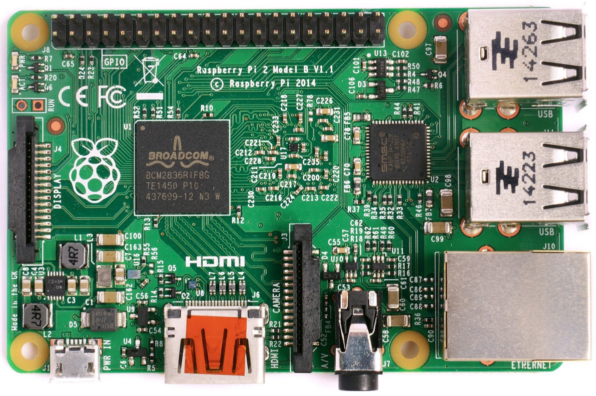


Figure 12: Raspberry Pi 2 B+

8.6.1 Reduce file system size

Because the new SD-card is only 4GB in size whereas the old one was 8GB, the file system must be shrinked before the card can be cloned to the new one.

Therefore the card is removed form the Pi and inserted into another Linux box. The card is recognized as /dev/sdf. Before shrinking the file system the partition is mounted and checked for the actual needed size:

```
#> mount /dev/sdf /mnt
#> df -h /mnt
#> umount /dev/sdf
```

According to the output only 3.2GB are used so we can now reduce the size of the file system accordingly:

```
#> fsck -f /dev/sdf2
#> resize2fs /dev/sdf2 3500M
```

8.6.2 Copy SD-card

Now the content oft he card can be copied over to the other card. Since there is only one card reader available the card is cloned to disk first and then copied over to the new card:

```
#> pv /dev/sd | dd of=pi.img bs=100M
#> pv pi.img | dd of=/dev/sdf bs=100M
```

The second command will fail with a “no space left on device” warning. That can be ignored since the file system was shrinked before so no needed content was lost.

8.6.3 Resize Partition

Now the data is on new card but the partition table is invalid. To repair this the second partition needs to be recreated. Therefore open fdisk for the device:

```
#> fdisk /dev/sdf
```

To bring up the manual press “**m**”.
We need to delete the second partition and recreate it. The new partition has to start at the same sector as the old one and has to be bigger then the files system (the default is to take up all the remaining space)

The raspi-config utility can now be used to expand the File System to use the whole partition:

```
#>raspi-config
```

8.6.4 Remove packages

To free up enough space for the needed update unnecessary packages should be removed:

```
#> aptitude purge \
idle python3-pygame python-pygame python-tk \
idle3 python3-tk \
python3-rpi.gpio \
python-serial python3-serial \
python-picamera python3-picamera \
debian-reference-en dillo x2x \
scratch nuscratch \
raspberrypi-ui-mods \
timidity \
smartsim penguinspuzzle \
pistore \
sonic-pi \
python3-numpy \
python3-pifacecommon \
python3-pifacedigitalio \
python3-pifacedigital-scratch-handler \
python-pifacecommon python-pifacedigitalio \
oracle-java8-jdk \
minecraft-pi python-minecraftpi \
wolfram-engine
#>apt-get clean
```

8.6.5 Upgrading system

Since the processor now belongs to the ARMv7 family whereas the previous used was from the ARMv6 family the operating system needs to be upgraded.

```
#> aptitude update
#> aptitude upgrade
#> rpi-update
```

9 Conclusion

Today web enabled end devices like smartphones are omnipresent. They offer a better screen and human interface than it is feasible or economical to integrate in a variety of embedded systems. By utilizing web enabled devices the user experience can be improved without adding additional production cost.
The hardware requirements on the backend side are not considerable higher then what is needed for a traditional, static web interface. As Linux is a popular choice for a lot of embedded devices there is also a large ecosystem available making backend development simple and straightforward.

However, development of a web application with proper user experience is a task, requiring a skillset that is very different from the skillset of an embedded developer. Also this technology is still very fast evolving and therefore needs constant attention. It may be possible for a single very skilled developer to keep up to both – web and embedded systems development – but this will – in most cases – not be economically.

For a product to feature a proper high quality web interface there should be a dedicated web developer. Depending on the complexity of the project even a dedicated user interface (UI) and user experience (UX) designer should be involved. If those resources are not available, outsourcing to a specialized agency should be considered.

Bibliography

[1] Official website of the AngularJS project – accessed 2015-5-30
https://angularjs.org/

[2] Official AngularJS developer documentation – accessed 2015-5-30
https://docs.angularjs.org/guide

[3] Official website of the Foundation project – accessed 2015-5-30
http://foundation.zurb.com/

[4] Official Foundation developer documentation project – accessed 2015-5-30
http://foundation.zurb.com/docs/

[5] Official website of the Angular-Foundation project – accessed 2015-5-30
http://pineconellc.github.io/angular-foundation

[6] Official website of the Bootstrap project – accessed 2015-5-30
http://getbootstrap.com/

[7] Official website of the Bootstrap project, component overview – accessed 2015-5-30
http://getbootstrap.com/components/

[8] IBM WebSphere Developer Technical Journal, Connect Ajax clients to near-real-time data with WebSphere Application Server Community Edition – accessed 2015-5-30
http://www.ibm.com/developerworks/websphere/techjournal/1008_sampathkumar/1008_sampathkumar.html

[9] General Websocket description – accessed 2015-5-30
https://www.websocket.org/

[10] Official website of the node.js project – accessed 2015-5-30
https://nodejs.org

[11] Official website of the Socket.io project – accessed 2015-5-30
https://socket.io/

[12] Official website of the Primus project – accessed 2015-5-30
https://github.com/primus/primus

[13] Official website of the Eureca.io project – accessed 2015-5-30
https://eureca.io/

[14] Official website of the PostgreSQL project – accessed 2015-5-30
http://www.postgresql.org/

[15] Official website of the MySQL project – accessed 2015-5-30
http://dev.mysql.com/

[16] Official website of the MariaDB project – accessed 2015-5-30
https://mariadb.org/

[17] Official website of the Redis project – accessed 2015-5-30
https://redis.io/

[18] Official website of the MongoDB project – accessed 2015-5-30
https://www.mongodb.org/

[19] Official website of Yeoman project – accessed 2015-5-30
http://yeoman.io/

[20] Github, generator-gulp-angular project page – accessed 2015-5-30
https://github.com/Swiip/generator-gulp-angular

[21] Official website of the GulpJS project – accessed 2015-5-30
http://gulpjs.com/

[22] Official website of the Bower project – accessed 2015-5-30
http://bower.io/

[23] General NTP description and resources – accessed 2015-5-30
http://www.ntp.org/

[24] elinux.org, How to install node.js on Raspbian – accessed 2015-5-30
http://elinux.org/Node.js_on_RPi

[25] Official website of the Raspberry Pi Foundation – accessed 2015-5-30
https://www.raspberrypi.org/

[26] Kofler.info, How to migrate from Raspberry to Raspberry 2– accessed 2015-5-30
https://kofler.info/raspberry-pi-2-alte-sd-karte-weiterverwenden/

[27] Wikipedia, picture of Raspberry Pi B – accessed 2015-5-30
http://de.wikipedia.org/wiki/Raspberry_Pi#mediaviewer/File:RaspberryPiModelBRev2.by.Philipp.Bohk.jpg

[28] Wikipedia, picture of Raspberry Pi 2 B+ – accessed 2015-5-30
http://de.wikipedia.org/wiki/Raspberry_Pi#/media/File:Raspberry_Pi_2_Model_B_v1.1_top_new.jpg

List of Figures

List of Abbreviations

ACID	Atomic, Consistent, Isolated, Durable
DOM	Document Object Model
E/R	Entity / Relationship
HDMI	High Definition Multimedia Interface
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
LAMP	Linux, Apache, MySQL, PHP
MVC	Model View Controller
NTP	Network Time Protocol
RDBMS	Relational DataBase Management System
REST	Representational State Transfer
RPC	Remote Procedure Call
SQL	Structured Query Language
UI	User Interface
URI	Unique Resource Identifier
URL	Unique Resource Location
UX	User eXperience
WWW	World Wide Web
XHR	XML HTTP Request