Progressive Web Apps: Performance, Offline Web and Home-Brewing

by Zac Colley

Submitted in partial fulfillment of the requirements for the award of the degree of BSc (Hons) Computer Science of the University of Portsmouth

April 2016

Preface

Dedicated to the Jesters and my family

Contents

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

Abstract

This project is based around performance and its relationship to user experience on the web. Performance will be explored through the design and implementation of a progressive web application for home-brewing.

Chapter 1

Introduction

This chapter introduces the topics and motivation for the project overall. The main goals will be discussed in section 1.1 followed by an outline of the chapters in this report.

1.1 Project Goals

The main goal of this project is to create a progressive web application (PWA). Progressive web applications are looking to take the best of the web and of native application development. Specifically some of the aspects of progressive web applications to be explored in the project are: Progressive enhancement of features and connectivity independent. (*Progressive Web Apps | Web Fundamentals - Google Developers*, n.d.)

Secondary goals including the research, experimentation and implementation of principles such as Performance as User Experience, offline first and modern web technologies such as Service Workers, Virtual DOM and clientside databases.

1.2 Report Outline

The rest of this report is organised as follows:

Chapter 2 reviews home-brewing and user experience through performance. Home-brewing is explained and compared to software. This is followed by an in depth discussion of user experience through performance, including techniques and methodologies to achieve performance.

Chapter 3 describes the design of the proposed solution. Topics in the section include: Discussion of tooling, design methodologies, sourcing and usage of data and choice of technologies. The requirements for the implementation are defined here.

Chapter 4 describes the system implementation of the proposed solution and testing. Based on the design this section discusses the implementation and interesting problems found.

Chapter 5 describes the testing and evaluation. The section includes description of the testing set-up, the results from testing and how well the requirements were fulfilled with examples.

Other overall evaluation is given here reflecting on the project as a whole.

Chapter 6 is the conclusion with future work. Here the report is wrapped up with some final thoughts.

Chapter 2

A Review of Home-brewing and User Experience through Performance

2.1 Introduction

This first section (2.2) of this chapter discusses the current landscape for home-brewing and it's integration with technology. The following section (2.3) explores the relationship between performance, user experience and modern web development. The analysis and design phase of this project will consider these topics. Finally, section 2.4 summarises the chapter.

2.2 Home-brewing

Last year, BBPA (The British Beer and Pub Association) estimated pubs in the UK had "fallen by 19% since 2000: from 60,800 to 49,433 in 2012". (*Research Briefings - Pub companies, pub tenants & pub closures: background history (up to 2014)*, n.d.)

Craft and independent brewing is on the up however. SIBA (The Society of Independent Brewers), saw an increase of SIBA's membership has grown by 49 members in 2015. (Cabras, n.d.)

Home-brewing, brewing beer on a small personal scale, has been around for thousands of years and has grown popular in the last few years.

2.2.1 Overview of a home-brew

(*How to Homebrew Beer*, n.d.) suggest there are four main ingredient and four main steps to brewing beer.

The four main ingredients of a brew are: **Malt**, **Hops**, **Yeast** and **Water**.

- **Malt** is a grain used to create the sugar for the yeast and determines how strong a beer is.
- **Hops** determines how bitter or sweet a beer is as well as other flavours and aromas.
- **Yeast** consume the sugars from the malt and convert them to alcohol and carbon dioxide (fizz). The yeast converts the wort (unfermented beer).
- **Water** makes up most of a brew by up to 90% by volume.

Taking these ingredients there are four steps to take: **Malting**, **Mashing**, **Boiling (and Cooling)** and **Fermenting (and Conditioning)**.

1. **Malting** starts the process of converting the ingredients into beer. This process results in dried grains by heating (or kilning).
2. **Mashing** soaks the malted grains for the previous step in hot water, this produces the maltose to feed the yeast. This also produces wort for the next step.
3. **Boiling (and Cooling)** the wort santises the brew, it's at this point the hops are added. After cooling the yeast is then added.
4. **Fermenting (and Conditioning)** starts to occur, this can last from weeks to months depending on the strength. Once this step is complete it's time to bottle the brew.

From the above process it is clear that this process is not trivial. Both beginner and expert home-brewers have sought out software to improved their brewing experience.

2.2.2 A comparison of home-brewing and OSS

Sharing recipes, set-ups and brewing techniques is analogous to the open source communities surround software and hardware. Bringing the two together is mutually beneficial.

With the popularity of affordable and accessible computing with platforms such as Raspberry PI and Arduino, introducing computing technologies into home-brewing means more automation and control.

Due to the open source nature of these platforms, many systems have been created. One example is BrewPi, a fermentation temperature controller for brewing beer or wine. (*BrewPi.com - The Raspberry Pi Brewing Controller*, n.d.)

This merge of physical creating and software can be seen all across the tech community. One great example is in farming with the MIT Open Agriculture Initiative, "Every time users grow and harvest, they will contribute to a library of climate recipes that can be borrowed and scaled so that users around the world can gain access to the best and freshest foods.". (*Climate Recipes*, n.d.)

Ultimately, the Internet is the perfect platform for this type of recipe sharing and collaboration.

2.2.3 Existing home-brewing solutions and APIs

Data formats have been created to handle brewing recipes and other brewing data, the most mature and popular of which is BeerXML. BeerXML uses the XML format for brewing data. (*BeerXML Recipe Standard*, n.d.) Open and standardized formats in general mean better portability and compatibility for data.

With a decline in usage of XML (*Why JSON will continue to push XML out of the picture - CenturyLink Cloud Developer Center*, n.d.), a JSON alternative has been created aptly named BeerJSON. (*BeerJSON - BeerJSON*, n.d.)

There are many software solutions for various parts of the brewing process. Some of which are reviewed in section 3.1.1.

One example is, Malt.io is a community website for these brewing recipes. With the options for revision history and 'cloning' of recipes it is clear it is inspired by the popular Git repository host and social coding network GitHub. (Taylor, n.d.) Git is discussed at length in section 3.1.3

2.3 User Experience through Performance

User experience (referred to as UX), is how a user feels while using a system. There are many aspects to user experience such as usability and accessibility. (*What Is User Experience Design? Overview, Tools And Resources – Smashing Magazine*, n.d.)

One of the most important aspects of user experience is the performance. When looking at the web specifically, performance in essence is the perceived speed of websites and the actions taken on them.

When designing and building on the web, performance should be considered at the same level as aesthetics. (*Performance is User Experience | Designing for Performance*, n.d.)

2.3.1 Speed and Timing

There is a lot of research into users' relationship with time.

(*Usability Engineering*, n.d.) shows time affects how users behave towards a system. A system feels instantaneous with actions taken under 0.1 seconds. Anything over 1 second a user can lose their flow of though on an action. After 10 seconds a users will lose attention and start to multi-task.

In terms of relative time, users perceive faster or slower tasks when there is a difference of 20% in time. (*Setting a performance budget - TimKadlec.com*, n.d.)

Google have "always viewed speed as a competitive advantage", when running experiments by introducing artificial delays into searches "slowing down the search results page by 100 to 400 milliseconds has a measurable impact on the number of searches per user of -0.2% to -0.6%". (*Research Blog: Speed Matters*, n.d.)

Performance delays aren't merely an annoyance, a bad first impression can result in users not returning. (*Why Web Performance Matters: Is Your Site Driving Customers Away?*, 2011) confirm that "88% of online consumers are less likely to return to a site after a bad experience". Delays are stressful. As shown by Figure 2.1, (*Ericsson Mobility Report*, n.d.) found that mobile delays are more stressful than standing in line at a shop or even watching a horror movie.

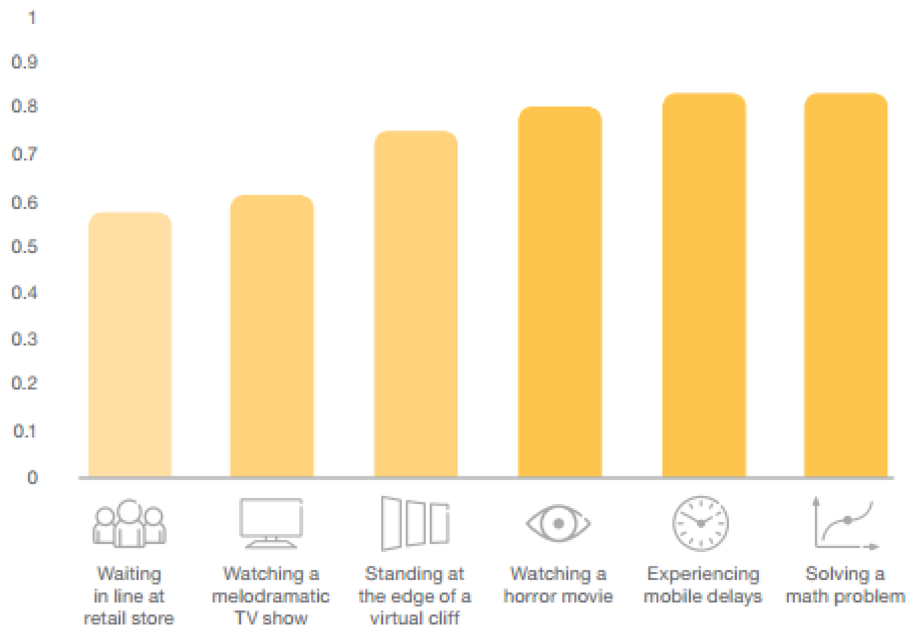


Figure 2.1: Cognitive load associated with stressful situations (*Ericsson Mobility Report*, n.d.)

There is a clear demand for performance when looking at the topic of blocking web advertising. A controversial topic as of writing, one of the benefits shown is the increase in performance for users especially on mobile platforms. Typically a browser extension would be used to block adverts. Brave, a browser built with advertisement blocking in from the start claims on page load 60% of the time is being caused by advertisement resources. (*Brave Software | Building a Better Web*, n.d.)

2.3.2 Perceived Performance

Before discussing achieving good performance through development aspects, it's worth noting a user is only perceiving this performance. This perceived performance often can be achieved without technical development.

Optimistic user interfaces (UIs) show a user feedback before the corresponding action has taken place thus giving an excellent perceived performance.

An example of this is the Instagram like button. On press of the like button the application doesn't wait for a round trip to the server to check that action was carried out successfully, instead it shows the user that they have liked the post immediately with the 'optimism' that the action will be carried out. (*LukeW | Mobile Design Details: Performing Actions Optimistically*, n.d.)

Lazy-loading is the method of loading content (often images) after initial load for performance. This is often used as part of an optimistic UI.

Using images as an example, Figure 2.2 shows the generation of a placeholder based on the source image for a smoother smoother transition. One technique to achieve this is to create a tiny thumbnail, let the browser resize and then blur to create a gradient based on the image's colour palette. (*Dominant Colors for Lazy-Loading Images | manu.ninja*, n.d.)

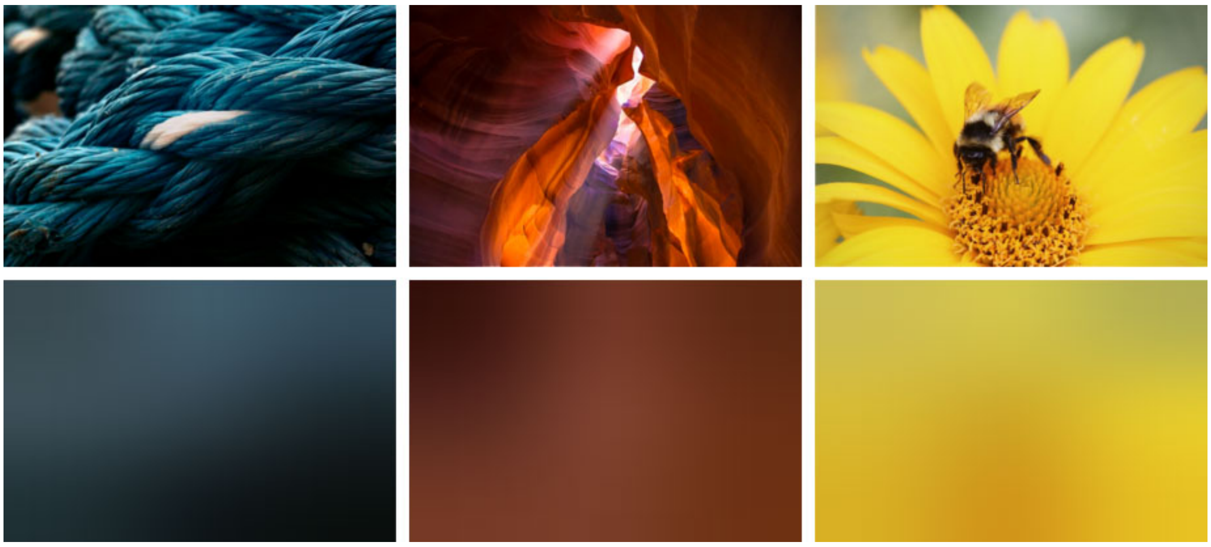


Figure 2.2: Example of placeholder when lazy loading (*Dominant Colors for Lazy-Loading Images | manu.ninja*, n.d.)

2.3.3 Measuring and techniques

Measuring performance is important, data metrics allow for tangible results. Tools such as Google's PageSpeed and WebPageTest give developers metrics across all aspects of web performance.

One metric is SpeedIndex which "is the average time at which visible parts of the page are displayed". The benefit to this metric is it helps measure a usable website over just a loaded website. A website may be usable without all of the content having loaded in. (*Speed Index - WebPagetest Documentation*, n.d.)

Best practices include optimising a website's resources and where they are retrieved from. With CSS file growing larger in size an emerging approach is through Critical CSS. Critical CSS is serving a part of the CSS on load, loading the rest asynchronously and then caching it all for further visits. (*How we make RWD sites load fast as heck | Filament Group, Inc., Boston, MA*, n.d.)

2.3.3.1 Performance Budgets

Having a performance budget can give a starting point for working with performance in mind. Having this initial constraint can ensure that performance will be thought of while designing. (Mall, n.d.)

One way to find initial metrics is off of similar websites. As described in section 2.3.1, setting the target for the metrics at 20% will mean users perceive the site as faster.

2.3.3.2 Front-end Frameworks and Libraries

Frameworks and libraries can improve development of websites but it is important to consider file size and overall performance of JavaScript when looking at performance.

Frameworks are generally expensive for any mobile devices, also there is a loss of control as to what final code is produced. (*Aerotwist - The Cost of Frameworks*, n.d.)

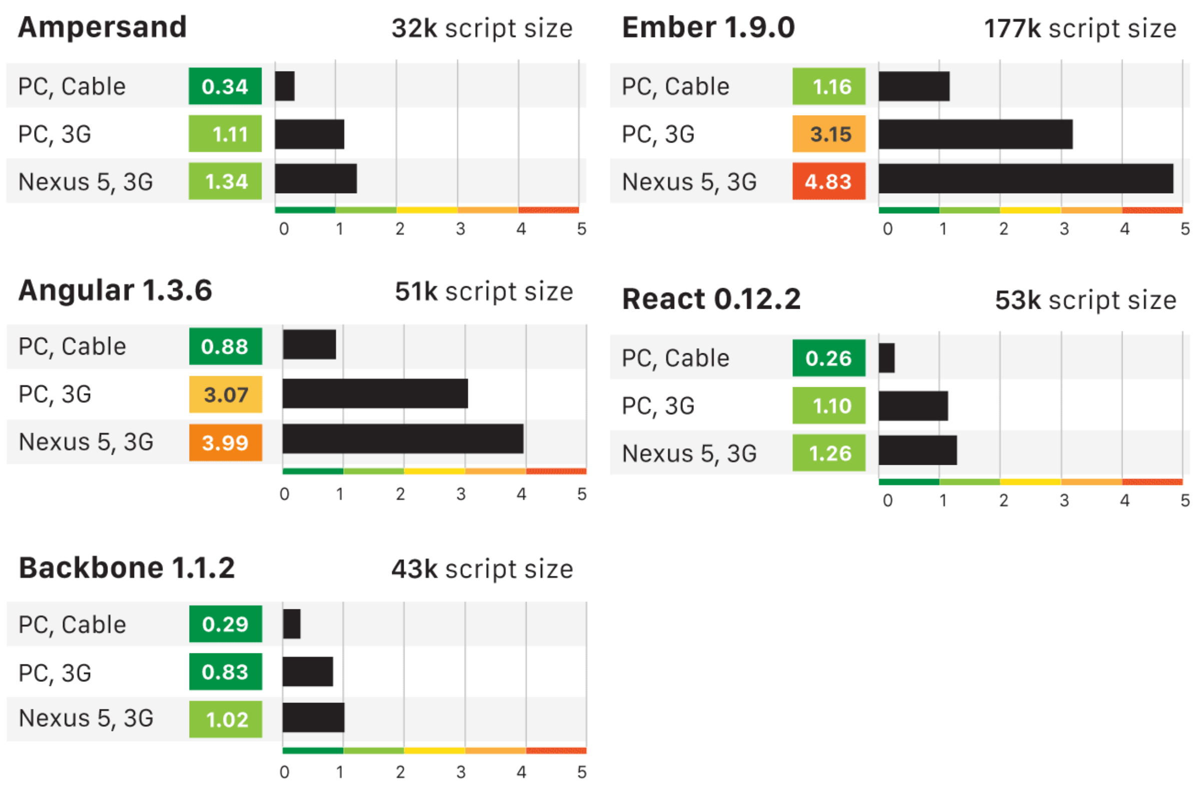


Figure 2.3: Average first render times (seconds) (Jehl et al., n.d.)

Figure 2.3 shows React has the quickest render on PC, with a cable network connection. Lightweight libraries like Backbone and Ampersand are also comparable to React's performance.

Angular with more features than React has a smaller size but performs much slower overall. Ember also with more features than React is both larger in file size and slower overall.

2.3.3.3 Animation

60 frames per second is the goal to hit when animating on the web due to most screens refreshing at this rate, if a site drops below this the screen judders commonly known as 'jank'. (*Rendering performance | Web Fundamentals - Google Developers*, n.d.)

When animating with CSS generally the properties to stick to are `transform` and `opacity`. These properties are hardware accelerated and therefore reduce jank. (*High Performance Animation - HTML5 Rocks*, n.d.) (*CSS Triggers*, n.d.)

Using the `will-change` property tells the browser there is a plan for it to animate. This is to replace using `translateZ()` as a hack to force the browser to hardware accelerate. (*Animations and performance | Web Fundamentals - Google Developers*, n.d.)

Interestingly, Microsoft Edge do not plan to implement `will-change` as they claim it is already optimised. (Storey, n.d.)

There are attempts to formalise these animation techniques into projects such as RAIL and LIP. (*Introducing RAIL: A User-Centric Model For Performance*, n.d.) (*Aerotwist - FLIP Your Animations*, n.d.)

2.3.4 Web typography

Good typography is a great way to improve user experience through usability and accessibility. For example, for some dyslexic users can find incorrectly spaced text hard to read through something called 'rivers'. (Maceri, n.d.)

Due to the popularity of web fonts it's easy to think this is a quick and easy way to improving the typographical user experience a website. However, just using a web font alone doesn't guarantee a better typography for users. Web fonts are slow, and if poorly implemented can leave users with no content at all on bad connections. This can create something called the Flash of Invisible Text (or 'FOIT') where the browser hides web font text until the web font has fully loaded. (*Font Loading Revisited with Font Events | Filament Group, Inc., Boston, MA*, n.d.)

A great reading experience with default system typefaces is easily achieved. As (*Webfonts - Writing*, n.d.) explains: "Typography is not about aesthetics, it's about serving the text. If even a small percentage of people don't consume your content due to a use of web fonts, your typography is failing."

There are ways to avoid FOIT through progressively applying fonts once they're loaded. These approaches work well for repeat visits to a website as these fonts can then be cached. (*Font Loading Revisited with Font Events | Filament Group, Inc., Boston, MA*, n.d.)

2.3.5 Progressive Enhancement

Progressive enhancement is treating extra layers of enhancement (such as images, dynamic behaviours) as optional. With this attitude, HTML is the only thing that should be relied upon. Sometimes even HTML can't reach the user (something discussed in section 2.3.6). (*Progressive enhancement — Government Service Design Manual*, n.d.)

There are arguments against progressive enhancement and that it can't always be achieved. There is a lot of discussion towards the progressive enhancement of JavaScript specifically. (*Progressive Enhancement: Zed's Dead, Baby : Tom Dale*, n.d.) claims that JavaScript can be assumed to be part of the web platform.

2.3.6 Offline First

"Frequently not having any data connection in even the wealthiest and most developed cities of the world has led us to conclude that no, the mobile connectivity/bandwidth issue isn't just going to solve itself on a global level anywhere in the near future." (*Say Hello to Offline First*, n.d.)

One of the biggest problems connecting to the Internet across the world is not the availability of devices but the network connections. Every year global mobile broadband subscriptions increase by 25%. In Q4 2015 there were 21 million in India, 21 million in Africa, 20 million APAC (excluding China and India). By comparison, 5 million Europe and 5 million in North America. (*Ericsson Mobility Report*, n.d.)

India's biggest e-commerce site Flipkart found 63% users reaching their mobile site via a 2G network. (*Flipkart Case Study*, n.d.) Developing with a stable and fast network connection in mind is not an option if considering these users.

A offline experience can be progressively enhanced with cached content managed by a Service Worker but even the poorest connection is online. Described by Jake Archibald: "Lie-Fi is like offline, but it trolls you by pretending to be online. It'll attempt to make a connection for minutes and still fail." (Google Developers, n.d.) This uncertainty with network connectivity means an site can never assume a user is completely offline.

Previous methods of caching content was through using Appcache, however the lack of control made it a very dangerous technology to use. (*Application Cache is a Douchebag · An A List Apart Article*, n.d.)

2.3.7 Workers

When developing on the web historically this meant running client-side Javascript in a single threaded environment. With the introduction of Workers web content can be handled with background tasks.

The two main Workers are Service Workers and Web Workers.

Web Workers create background tasks that can run Javascript that interface back and forth between the main JavaScript tasks. A use case of Web Workers is syncing data between a clientside database and a remote database. (*Using Web Workers - Web APIs | MDN*, n.d.)

Service Workers can be seen more as a proxy server between the server and the main JavaScript thread, the key concept is that of 'installing' a Service Worker file to the browser itself registered to a origin. Service Workers are asynchronous and so don't have access to APIs such as XHR.

Some use cases of Service Workers include, caching for a better (and sometimes offline) experience, Push notifications, background data syncing. (*Service Worker API - Web APIs | MDN*, n.d.)

2.3.8 Performance based projects

Google have set-up the Accelerated Mobile Pages (AMP) Project as a way to create content optimized for mobile. This project grew from a discussion "between publishers and technology companies about the need to improve the entire mobile content ecosystem for everyone". AMP is constrained to ensure reliable performance, at the cost of limited flexibility in content. (Google Chrome Developers, n.d.)

2.4 Summary

In Chapter 1 we proposed user through performance with the example of home-brewing.

In this chapter home-brewing was introduced (section 2.2) followed by an extensive review of user experience through performance (section 2.3).

The next chapter presents the design of a progressive home-brewing based web application following the discussion from this chapter.

This page is intentionally left blank.

Chapter 3

Artefact Design

3.1 Introduction

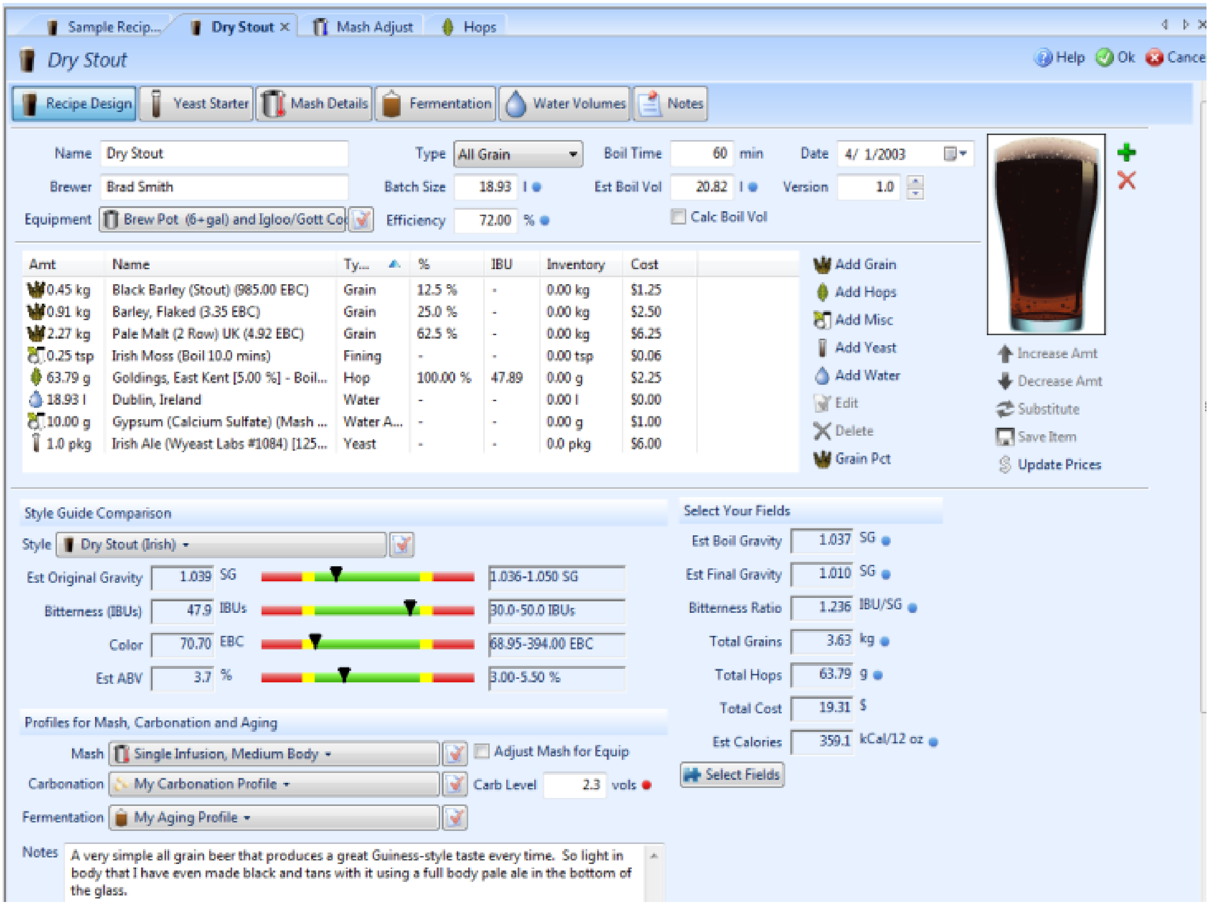
In Chapter 3 we identified that one of the biggest aspects of user experience is performance and that home-brewing is a complex area to try and improve through software.

This chapter describes the design of a progressive web application with offline functionality for the use in a home-brewing context. First, the chapter reviews existing software in home-brewing (section 3.1.1), tools and service to used in the implementation (section 3.1.3) and system requirements (section 3.2). A proposed solution is then discussed (section 3.3) including initial designs and technologies.

3.1.1 A Review of Existing Software

This section is a review of current home-brewing software.

3.1.1.1 BeerSmith



Dry Stout

Name	Dry Stout	Type	All Grain	Boil Time	60 min
Brewer	Brad Smith	Batch Size	18.93 l	Est Boil Vol	20.82 l
Equipment	Brew Pot (6+gal) and Igloo/Gott Co	Efficiency	72.00 %	Calc Boil Vol	☐
Date	4/ 1/2003	Version	1.0		

Amt	Name	Ty...	%	IBU	Inventory	Cost
0.45 kg	Black Barley (Stout) (985.00 EBC)	Grain	12.5 %	-	0.00 kg	$1.25
0.91 kg	Barley, Flaked (3.35 EBC)	Grain	25.0 %	-	0.00 kg	$2.50
2.27 kg	Pale Malt (2 Row) UK (4.92 EBC)	Grain	62.5 %	-	0.00 kg	$6.25
0.25 tsp	Irish Moss (Boil 10.0 mins)	Fining	-	-	0.00 tsp	$0.06
63.79 g	Goldings, East Kent [5.00 %] - Boil...	Hop	100.00 %	47.89	0.00 g	$2.25
18.93 l	Dublin, Ireland	Water	-	-	0.00 l	$0.00
10.00 g	Gypsum (Calcium Sulfate) (Mash ...	Water A...	-	-	0.00 g	$1.00
1.0 pkg	Irish Ale (Wyeast Labs #1084) [125...	Yeast	-	-	0.0 pkg	$6.00

Style Guide Comparison

Style: Dry Stout (Irish)

Est Original Gravity	1.039	SG	1.036-1.050 SG
Bitterness (IBUs)	47.9	IBUs	30.0-50.0 IBUs
Color	70.70	EBC	68.95-394.00 EBC
Est ABV	3.7	%	3.00-5.50 %

Select Your Fields

Est Boil Gravity	1.037	SG
Est Final Gravity	1.010	SG
Bitterness Ratio	1.236	IBU/SG
Total Grains	3.63	kg
Total Hops	63.79	g
Total Cost	19.31	$
Est Calories	359.1	kCal/12 oz

Profiles for Mash, Carbonation and Aging

Mash	Single Infusion, Medium Body
Adjust Mash for Equip	☐
Carbonation	My Carbonation Profile
Carb Level	2.3 vols
Fermentation	My Aging Profile

Notes: A very simple all grain beer that produces a great Guiness-style taste every time. So light in body that I have even made black and tans with it using a full body pale ale in the bottom of the glass.

Figure 3.1: A screenshot of the BeerSmith's interface

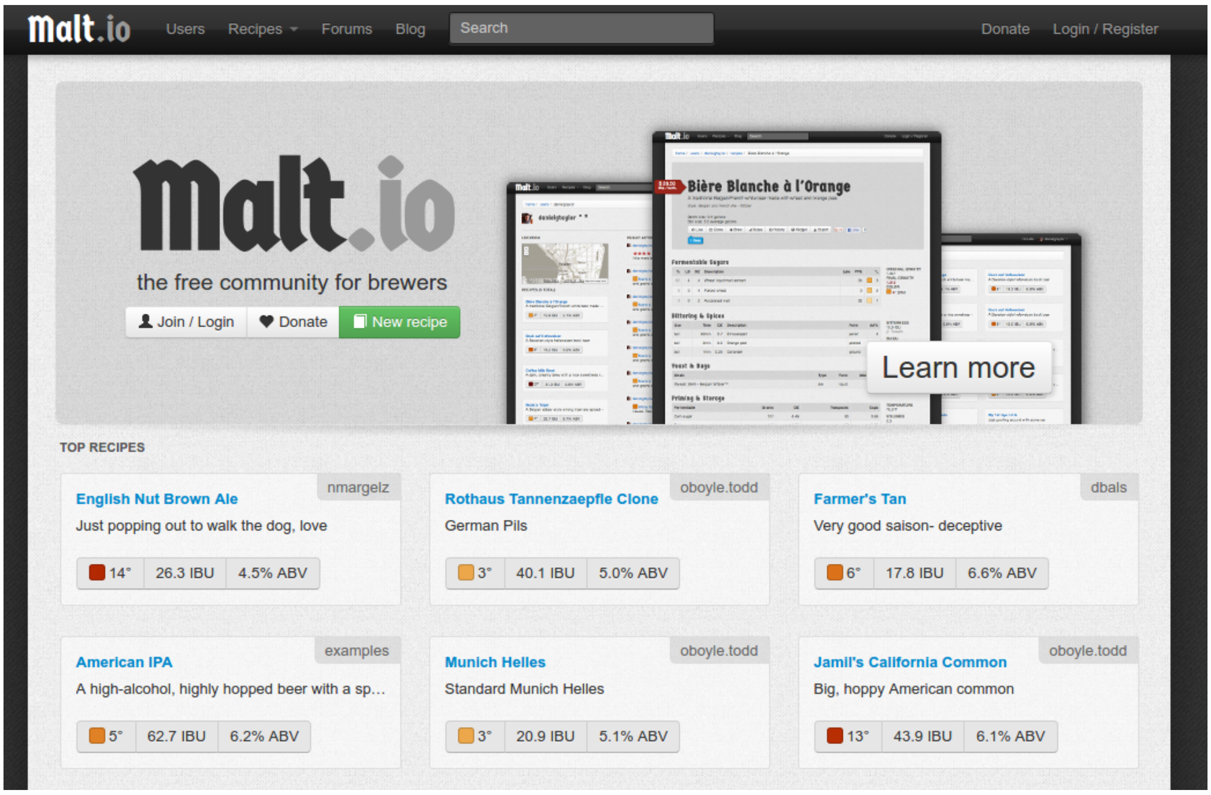
Positive

- ✓ Cross platform (Macintosh, Ubuntu and Windows)
- ✓ Offline by nature as it's installed to the computer
- ✓ Allows timing of brews and taking notes

Negative

- ☐ Complex interface with overwhelming features for beginners
- ☐ Simplistic sharing of recipes

3.1.1.2 Malt.io



TOP RECIPES

English Nut Brown Ale	nmargelz	14°	26.3 IBU	4.5% ABV
Just popping out to walk the dog, love				
Rothaus Tannenzaepfle Clone	oboyle.todd	3°	40.1 IBU	5.0% ABV
German Pils				
Farmer's Tan	dbals	6°	17.8 IBU	6.6% ABV
Very good saison- deceptive				
American IPA	examples	5°	62.7 IBU	6.2% ABV
A high-alcohol, highly hopped beer with a sp…				
Munich Helles	oboyle.todd	3°	20.9 IBU	5.1% ABV
Standard Munich Helles				
Jamil's California Common	oboyle.todd	13°	43.9 IBU	6.1% ABV
Big, hoppy American common				

Figure 3.2: A screenshot of the Malt.io's home page

Positive

✓ Available on any platform with a web browser

✓ Social network style features (profiles, sharing)

✓ History and cloning of recipes

Negative

☐ No offline functionality

☐ Lack of timeline functionality

One thing that is common among these choices is their inherent offline nature. As they are installed directly to the computer and not in a web browser.

As seen in section 3.2, this review of existing software will be used to influence the proposed solution's design.

3.1.2 Software Life-cycle Methodology

Choosing a methodology is never one size fits all. Instead considering the environment and team around a project is important. Agile software development offers principles that work well with a team, however there is a lot that can be taken for any project.

Agile development embraces changes in development life-cycle and adapts to them, this is in contrast to traditional plan-driven development in which predictions are used instead. An agile plan is the initial set of goals which will adapt as the project grows. (Fowler, n.d.)

Using an incremental and evolutionary approach is key for this artefact due to its experimental nature, the goals and proposed solutions will change through-out development.

3.1.3 Tools and Services

Kanban boards are a tool to visualise the workflow for a project, an online implementation of this tool is Trello. Trello can be used for multiple scenarios and fits well for a software project. Features, bugs and other research can be represented by cards. (*Trello*, n.d.)

Version control is arguably one of the most important tools in software development. It records changes over time so that a previous version of the project can be revisited later. Git is one of the most popular version control software currently. (*Git - About Version Control*, n.d.)

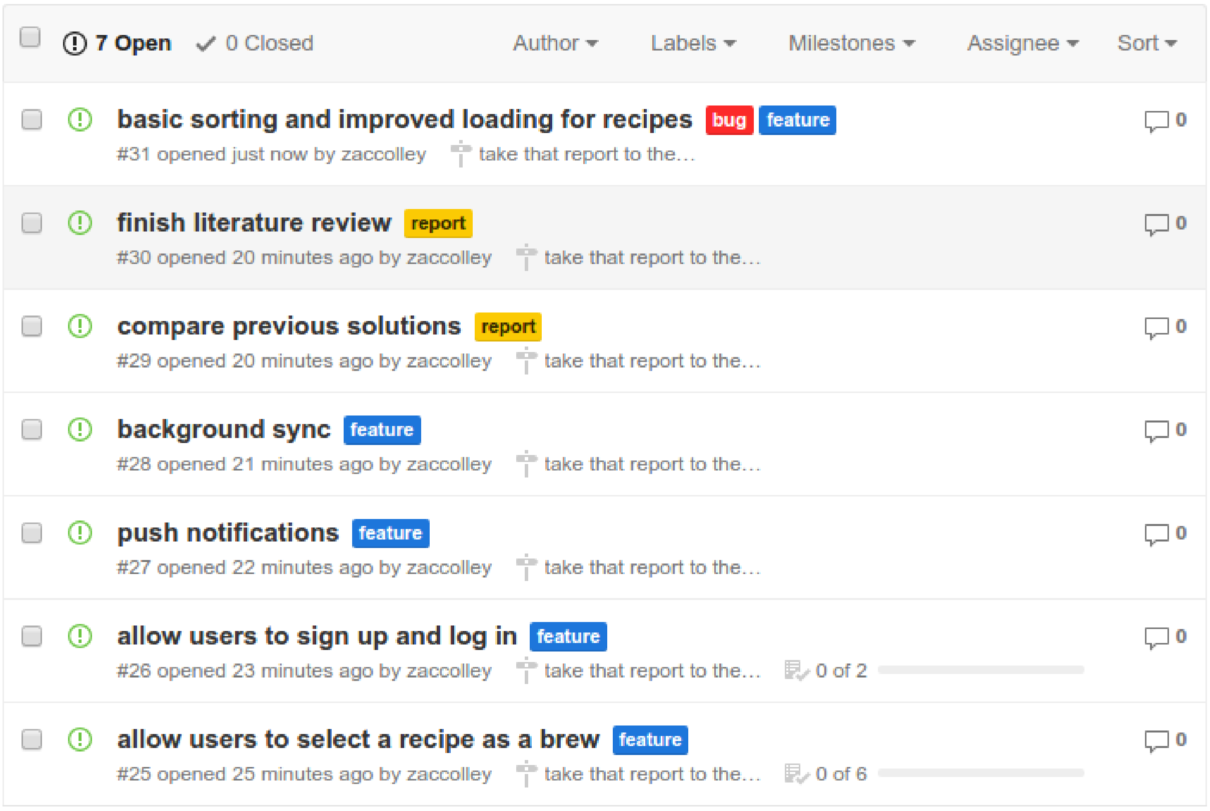
While being distributed it's often useful to have a centralized repository, there are many Git repository hosting services. GitHub and Bitbucket are two popular choices. While GitHub encourages Open Source Software on their platform the service itself isn't Open Source. (*GitHub*, n.d.) GitLab started as a GitHub clone that is fully Open Source, with the choice to host independantly or use their servers. (*Code, test, and deploy together with GitLab open source git repo management software | GitLab*, n.d.) (*Bitbucket*, n.d.)

These platforms offer similar features but the artefact will be hosted with GitHub due to the familiarity to the author and also the features it offers.

GitHub has a system of 'Issues', which allow for tasks to be tracked. These issues can be commented and then closed when deemed completed. For this project the following labels are used:

- **bug** - tasks which define a bug in the project
- **feature** - tasks which introduce a new feature to the project
- **report** - tasks which are relevant to the report writing itself
- **greenkeeper** - a label reserved for the greenkeeper tool (as described in section 4.5)
- **wontfix** - tasks that are out of scope and/or outdated and won't be completed

Figure 3.3 shows a screenshot of the GitHub issues at one point in the project.



basic sorting and improved loading for recipes bug feature
#31 opened just now by zaccolley take that report to the… 0

finish literature review report
#30 opened 20 minutes ago by zaccolley take that report to the… 0

compare previous solutions report
#29 opened 20 minutes ago by zaccolley take that report to the… 0

background sync feature
#28 opened 21 minutes ago by zaccolley take that report to the… 0

push notifications feature
#27 opened 22 minutes ago by zaccolley take that report to the… 0

allow users to sign up and log in feature
#26 opened 23 minutes ago by zaccolley take that report to the… 0 of 2 0

allow users to select a recipe as a brew feature
#25 opened 25 minutes ago by zaccolley take that report to the… 0 of 6 0

Figure 3.3: Example of GitHub issues setup

3.2 Requirements

Both the supervisor Rich Boakes and local Portsmouth home-brewer Alan Thompson of Get-Brewing.uk, fueled the initial requirements for this project. They saw a need for applications that help improve the current home-brewing ecosystem.

As follows are requirements to create such an application to improve the home-brewing process when following recipes. After reviewing the software available in section 3.1.1 the proposed solution can be influenced by their features:

- Web based project for the inherent ability to share easily
- Offline accessibility
- Easy to use and focussed feature set

The world wide web was initially developed to share documents, this is still applicable now and makes for a great platform for sharing features in this environment.

One key aspect from a lot of this software is as it is installed to a computer it is offline. Something a traditional web application couldn't achieve. Having the artefact introduce offline features makes for an innovative solution.

BeerSmith chooses to aim at a expert user, Malt.io is more general but still offers many options. A simpler and focussed feature set will be used for the proposed solution for these reason but also in the interest of keeping the scope of the project achievable.

3.2.1 Functional Requirements

The following requirements will be structured through user stories.

3.2.1.1 Finding recipes

- As a **user**, I want to **browse for recipes**

3.2.1.2 Starting a brew

- As a **user**, I want to **select and start a brew from any recipe**

3.2.1.3 Managing a brew

- As a **user**, I want to **be shown what point I am in a brew**
- As a **user**, I want to **add notes to key stages of a brew**
- As a **user**, I want to **review and evaluate the brew on completion**

3.2.2 Non-Functional Requirements

The artefact has the following non-functional requirements.

- Compliant to WCAG 2 level AAA to ensure enough contrast "when viewed by someone having color deficits or when viewed on a black and white screen". (*Techniques For Accessibility Evaluation And Repair Tools*, n.d.)
- Be documented so that others can maintain
- Conform to the performance budget that is set in the proposed solution
- Pass tests generated throughout and running through continuous integration (such as Travis CI)

3.3 Proposed Solution

Figure 3.4, shows an overview of the application stack to be used in the proposed solution. The following sections will describe different parts of this diagram and more.

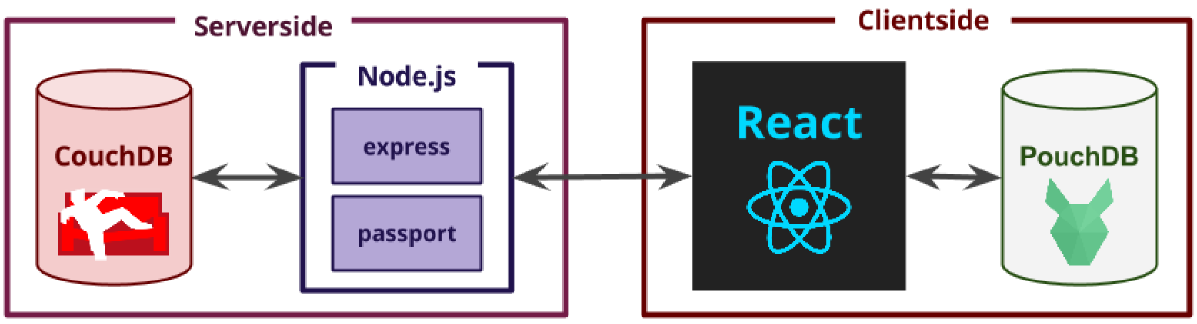


Figure 3.4: Overview of how the application stack is structured

3.3.1 User Interface Wireframes

While the user interface isn't the focus of this project, some design here is important to work towards the simple interface for overall usability.

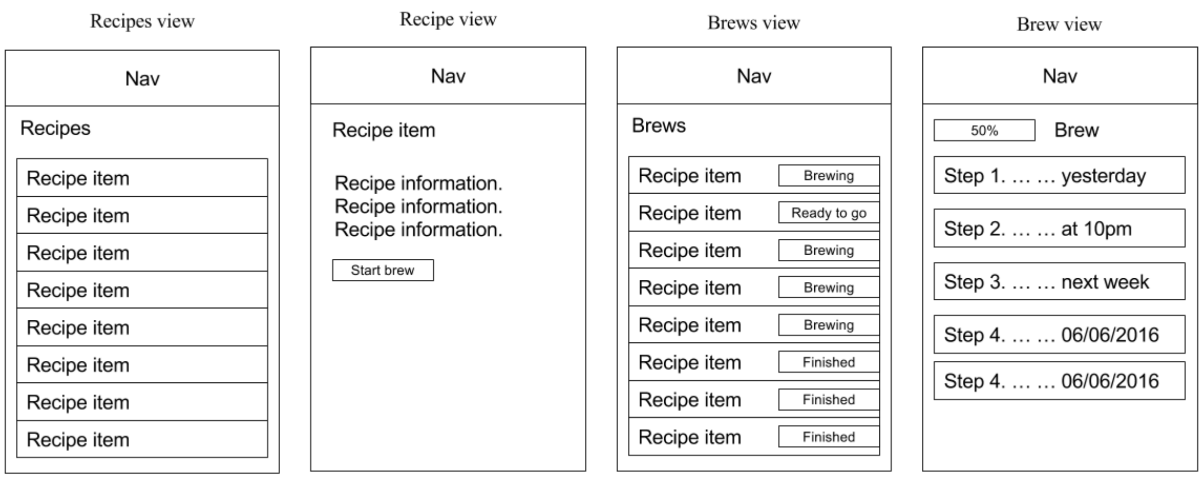


Figure 3.5: Wireframes for the main views

3.3.2 Performance Budget

As an initial target for the performance budget we will use a similar website speeds and try and beat them by 20% (as explained in section 2.3.3.1).

The chosen website is Malt.io, which has similar functionality to the proposed solution.

Site	**Start Render**	**Document Complete**	**Fully Loaded**
Malt.to	2.190s	3.998s	4.145s
Target	1.752s	3.198s	3.316s

Table 3.1: Performance budget calculation

The proposed solution will aim to have timings that are equal or below to the results in

Table 3.1.

With performance in mind initially the project will not implement any web fonts or images. The recipes and brew instructions have been determined as the main content and so will be the main focus.

Any iconography will use SVGs for flexibility and overall performance.

3.3.3 Data

This sub-section will describe the gathering, creation and storage of data for the proposed solution.

3.3.3.1 Recipe Gathering

As noted in section 2.2.3, BeerXML is a popular format. The proposed solution will scrape (process pages using scripts) existing recipe lists for BeerXML recipes.

These files will then be converted into BeerJSON which will be the main recipe format used.

3.3.3.2 Databases

Due to the format of choice BeerJSON, a document store style database that deals directly with JSON documents is a great choice.

There are many document store databases to choose from for example MongoDB, however the chosen database for the proposed solution is CouchDB for its HTTP API and master-to-master replications. (*Apache CouchDB*, n.d.)

The artefact can also make use of PouchDB, a JavaScript database which leverages IndexedDB for clientside databases. PouchDB also has the same API as CouchDB allowing them to sync seamlessly.

An alternative to PouchDB would be localForage. localForage, which also allows for offline databases using IndexedDB, is simpler than PouchDB however the syncing feature is what makes PouchDB the chosen technology.

The proposed solution will attempt to use a Web Worker for the syncing and management of data, this will free up the main JavaScript file for presentation tasks.

3.3.4 Technologies

Traditionally JavaScript has been in the browser. Node.js allows for using JavaScript on the server using Chrome's V8 JavaScript engine. (*Node.js*, n.d.)

npm is a package manager for Node.js (more generally JavaScript). npm will be used for both serverside and clientside JavaScript dependencies.

The latest version of JavaScript ES2015 (also known as ES6) offers features that will be used in the proposed solution such as class structure for React components. Support for ES2015 is mixed across browsers, therefore converting (transpiling) ES2015 to ES5 will be carried out with Babel a JavaScript transpiler. (*Babel · The compiler for writing next generation JavaScript*, n.d.)

3.3.4.1 Serverside

In this proposed solution three main packages on the serverside will be: Express, Passport and Brahaus. (*npm*, n.d.)

Express is a framework which offers features such as path routing and templating. In the proposed solution this package will be used to handle requests from the client to the server and some basic data manipulation from the database. (*Express - Node.js web application framework*, n.d.)

Passport is a framework for authentication, including users and user sessions. In the proposed solution this package will allow for users to sign-up, log-in and have a session in the application. (*Passport*, n.d.)

Due to the artefact's knowledge base around home-brewing, a package called Brauhaus will be used. Brauhaus handles conversions of brewing recipes and brewing information. Using this package means that the home-brewing knowledge such as calculations is abstracted away, meaning developers don't necessarily need to fully understand the knowledge base to create the application. (*Malt.io Blog: Introducing Brauhaus.js*, n.d.)

3.3.4.2 Clientside

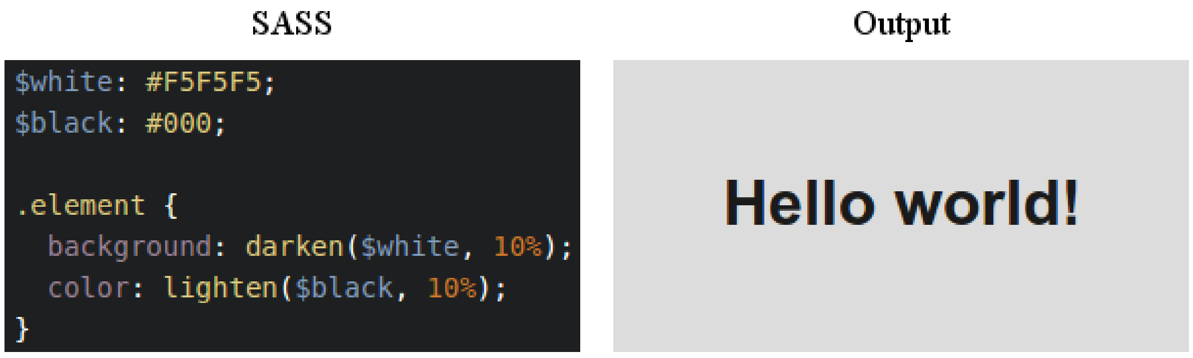
From the review of frameworks in section 2.3.3.2, the proposed solution will use React as it appears to offer the best balance of features to performance.

React's usage of JSX, a syntactic sugar for improving the creation of components will be used throughout the project. SVGs will be used for graphics, this is complimentary as both SVG and JSX. (*JSX in Depth | React*, n.d.)

Using a Service Worker a cache will be created to enable files to be retrieved even when offline.

3.3.4.3 Styling

For styling various technologies will be used to process into the final CSS stylesheet. CSS preprocessors convert a higher level CSS extension language with features such as variables and methods into CSS.



```
$white: #F5F5F5;
$black: #000;

.element {
  background: darken($white, 10%);
  color: lighten($black, 10%);
}
```

Figure 3.6: Example of SASS variables and methods to manipulate colours

There are many pre-processors for CSS, such as LESS, SASS or Stylus. The proposed solution will be using SASS however, due to only using the basic features being used other preprocessors could be used. (*Sass: Syntactically Awesome Style Sheets*, n.d.)

Styling often comes with some pitfalls, such as browser prefixes for CSS properties which are not stable yet. A post-processor such as Autoprefixer adds these prefixes automatically. (*postcss/autoprefixer: Parse CSS and add vendor prefixes to rules by Can I Use*, n.d.)

'postcss' will be used to transform these styles in the JavaScript environment. (*postcss/postcss: Transforming styles with JS plugins*, n.d.)

3.3.5 Workflow

As JavaScript is dynamic and loosely-typed, using a linter can ensure certain common problems are avoided. ESLint will be the linter used in the proposed solution due to its extensibility and features for JSX. (*About - ESLint - Pluggable JavaScript linter*, n.d.)

Linting will be carried out with continuous integration (see section 3.3.5.1) but also inside IDEs and code editors on save of JavaScript documents. For this reason Atom, a code editor, will be used for it's ESLint plugin.

There are many task runners for improving a development workflow such as Gulp, Grunt and Broccoli. A technology that will be used in place of task runners for this artefact is Webpack. Webpack bundles all dependencies that are required by the application and indexes them. Webpack knows what dependencies to bundle into the final file whereas Gulp and Grunt have to be told which dependencies to bundle through configuration. This results in a smaller file. (*webpack module bundler*, n.d.)

A smaller file will help for the proposed performance budget and so Webpack will be used. Other technologies such as rollup, which uses 'tree-shaking' to achieve a smaller bundle than Webpack can be explored but due to it's infancy this will seen as an experimental implementation. (*rollup/rollup*, n.d.)

Webpack will be used to transpile code using Babel, process Sass, process JSX from React and more.

npm will be used for clientside dependencies. Due to the popularity of projects such as Browserify, which create browser friendly versions of packages, it is used for clientside as well as serverside JavaScript. (*Browserify*, n.d.) Managers such as bower which dedicated itself to clientside dependencies have seen a drop in usage due to this. (*Bower*, n.d.)

Due to the nature of npm, dependencies depending on many different dependencies. Tracking security on dependencies is a daunting task. Snyk is a package which finds, patches and monitors vulnerabilities in code. (*Snyk*, n.d.) Snyk tests will ensure there isn't a deployed version of the build with known vulnerabilities.

3.3.5.1 Continuous Integration and Deployment

As ThoughtWorks describes "Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.". (*Continuous Integration | ThoughtWorks*, n.d.)

There are different ways to achieve CI, the artefact will use Travis CI. Travis CI is a online software-as-a-service CI platform. It allows developers to define a set of tasks to carry out (through a `.travis.yml` file in the repository). These tasks should have a success and fail conditions for Travis to know the status of a build. Examples of tasks include building from dependences, running tests and even miscellaneous tasks such as processing LaTeXdocuments. [1]

The deployment will also be triggered by these tests succeeding, ensuring unstable builds of the application are never deployed.

Surge (surge.sh) will be used to deploy, it is command-line tool that allows for simple deployment. Installed through npm, it removes any need for FTP, SSH or other more complicated deployment set-ups. Obviously this has limitations, but for clientside web applications this is a simple and effective option. (*Surge*, n.d.)

This CI is triggered by pushing to the GitHub repository. This allows for both the code, the tests and deployment to all be together.

3.4 Summary

This chapter described the high-level requirements and design of a progressive web application for home-brewing. The chapter started by the existing software used for home-brewing. The proposed solution was then discussed in following sections.

[1] This report was created using LaTeX, a word processing and document markup language. (*LaTeX – A document preparation system*, n.d.) This report uses ShareLatex's CI to automatically build the final report PDF when the LaTeXfiles are pushed to the GitHub repository. Magic! (*Github LaTeX CI*, n.d.)

The implementation based on this design is covered in further detail in Chapter 4 which describes the system implementation.

Chapter 4

System Implementation

4.1 Introduction

This chapter describes how the system was implemented based on the previous artefact design. From design to implementation a few decisions and technologies changed.

Figure 4.1, shows the revised application stack for the implementation.

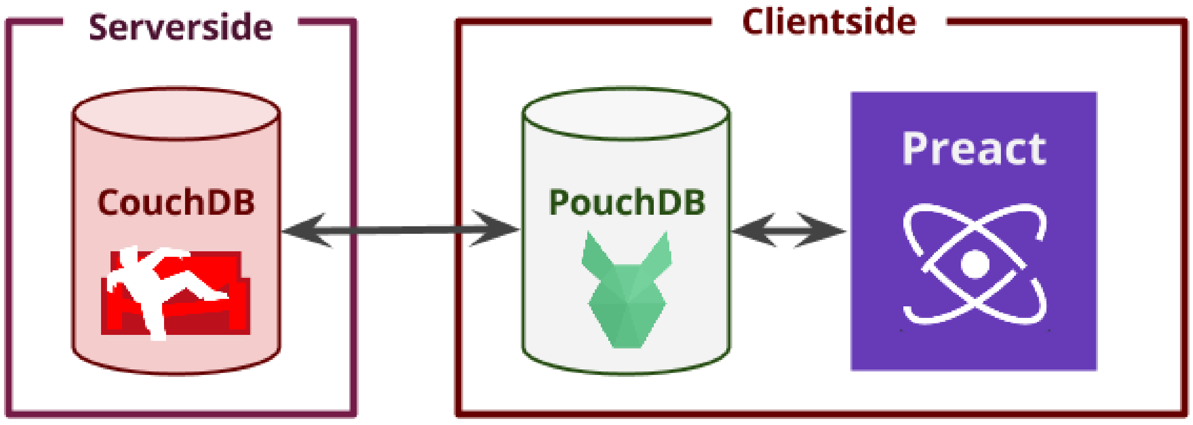


Figure 4.1: Overview of how the application stack is structured for the implementation

The biggest change was the removal of Node.js stack including Express and Passport. As explained fully in section 4.4 the data manipulation and authentication functionality were handled through CouchDB.

4.2 Packages

During implementation and research into React and performance, another framework called Preact was discovered. Preact, is a framework that attempts to recreate functionality of React but with the focus of performance. Using Preact means a better development experience. (*developit/preact*, n.d.) Preact is 3kb in size in comparison to React which is 53kb in size.

The implementation therefore used Preact and was used for rendering views in the artefact.

One principle for software development is DRY (don't repeat yourself). One principle for software development is DRY (don't repeat yourself). "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." (*Dont Repeat Yourself*, n.d.)

A good example of this is the 'moment.js' package used in this artefact, this package handles most date and time related problems. To implement these features is not inherently difficult, but packages such as moment.js offer DRY tried and tested solutions. (*Moment.js | Home*, n.d.)

4.3 Service Worker and Offline Functionality

Offline caching of files was implemented through service worker. At the time of writing, there is discussion as to whether service worker is too low level for using the caching functionality. It is argued that this functionality could be brought through as a separate caching API. (*Service Workers replacing AppCache: a sledgehammer to crack a nut — Medium*, n.d.)

Implementing the caching with service worker wasn't trivial but once learnt was understandable. Having an abstracted caching API could be useful for simpler implementations of offline caching.

Having a clientside database through PouchDB and IndexedDB (as further discussed in section 4.4) not only means offline data but also quick retrieval of data even when connected to a network. This is a boost for performance in general.

4.4 Data and Databases

As previously mentioned CouchDB is excellent at master-to-master replication of databases. This means syncing is easily implemented.

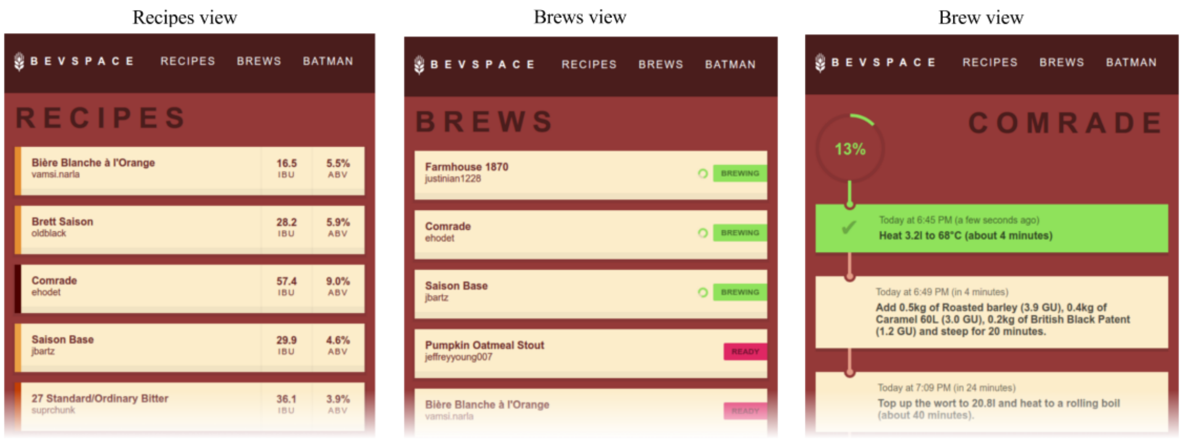
The remote database and the local databases sync any changes between themselves, when the local database changes the recipes and brews in state are updated.

In the artefact design data section 3.3.3 and following the decision was to use Node for a Express to handle data coming from the database and Passport to handle user authentication. However through the use of CouchDB and PouchDB it was found that there was no need for either of these technologies.

CouchDB has a user authentication system for handling permission and roles to documents. By default everyone is an admin in what they describe as the 'admin party', this is recommended to be disabled but aids development with low barrier to entry when starting a new project.

As the data defines a lot of the structure of the application this authentication system has been leveraged for user authentication for the application.

4.5 Interesting Problems



RECIPES

Bière Blanche à l'Orange vamsi.narla	16.5 IBU	5.5% ABV
Brett Saison oldblack	28.2 IBU	5.9% ABV
Comrade ehodet	57.4 IBU	9.0% ABV
Saison Base jbartz	29.9 IBU	4.6% ABV
27 Standard/Ordinary Bitter suprchunk	36.1 IBU	3.9% ABV

BREWS

Farmhouse 1870 justinian1228	BREWING
Comrade ehodet	BREWING
Saison Base jbartz	BREWING
Pumpkin Oatmeal Stout jeffreyyoung007	READY
Bière Blanche à l'Orange vamsi.narla	READY

COMRADE

13%

Today at 6:45 PM (a few seconds ago)
Heat 3.2l to 68°C (about 4 minutes)

Today at 6:49 PM (in 4 minutes)
Add 0.5kg of Roasted barley (3.9 GU), 0.4kg of Caramel 60L (3.0 GU), 0.2kg of British Black Patent (1.2 GU) and steep for 20 minutes.

Today at 7:09 PM (in 24 minutes)
Top up the wort to 20.8l and heat to a rolling boil (about 40 minutes).

Figure 4.2: Screenshots of different views as implemented

During the implementation of this project several issues came up. This section discusses some of these issues.

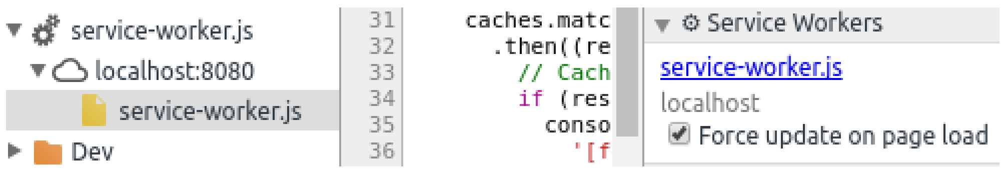
Using newer technologies such as Webpack, ESLint, Babel and Preact come with difficulties. There is a generally a lack of support found online and even some issues can be so new they haven't been documented yet.

For example, with Preact during implementation there was an issue with a feature from React not being supported. This was short-circuit boolean templating, the author then reported this bug which was then fixed. Here having the Preact project as an Open Source project meant this could be fixed not just for this artefact but future users of Preact.

While testing the service worker code and implementation difficulties arise when caching files. Originally the author was manually unregistering the service worker then closing and reopening the tab.

Simply refreshing does not give any updated version of the service worker files, this is due to the browser making the new page request before unloading the current page so the activated service worker is never released. (*Using ServiceWorker in Chrome today - JakeArchibald.com*, n.d.)

As shown in Figure 4.3, to solve this problem there is a 'Force update on page load' option in the Chrome development tools.



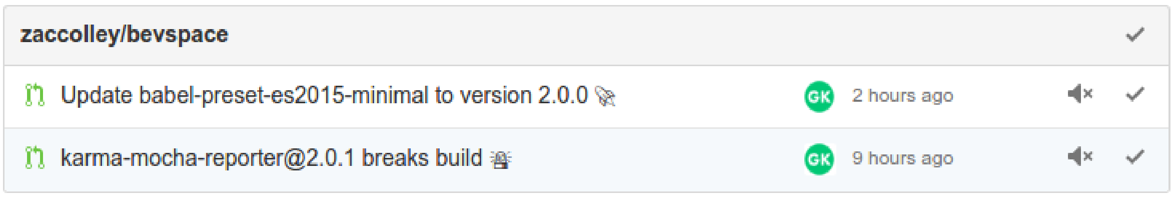
```
caches.matc
  .then((re
    // Cach
    if (res
      conso
      '[f
```

service-worker.js
localhost
☑ Force update on page load

Figure 4.3: The ‘force update on page load’ option

One issue with frameworks and libraries expressed in section 2.3.3.2 was the lack of control of final code. Having to depend on the Brauhaus package due to a lack of home-brewing knowledge resulted in having to implement more of the library than potentially needed and ultimately a bigger file size reducing performance.

Keeping dependencies up to date is important mostly for security (see Snyk mentioned in section 5.2), a tool called Greenkeeper (shown in Figure 4.4) was used through-out which monitors the project’s dependencies and sends a Pull Request to the GitHub repository with an updated version of packages. This is great as these Pull Requests can be run through the continuous integration and if it passes the build can be merged straight away. (*Index | Greenkeeper*, n.d.)


zaccolley/bevspace

Update babel-preset-es2015-minimal to version 2.0.0 — GK — 2 hours ago

karma-mocha-reporter@2.0.1 breaks build — GK — 9 hours ago

Figure 4.4: Example of Greenkeeper testing different package updates

One problem when developing with service workers (and other web APIs) is needed SSL or https when in production. Thankfully, `localhost` is ignored so these features can be used in development easily. Also with projects such as Let’s Encrypt, adding SSL to a web site is free and easy. (*Let’s Encrypt - Free SSL/TLS Certificates*, n.d.)

4.6 Summary

This chapter described the change in technology stack, interesting problems surrounding working with newer technologies such as service worker. It also discusses tools and packages used for the implementation and their benefits.

Chapter 5

Testing and Evaluation

5.1 Introduction

Chapters 3 and 4 described the design and implementation of a progressive web application for use in a home-brewing context. In this chapter, the testing method is discusses with results in section 5.2. The next chapter evaluates the implementation as a whole in section 5.3. This is followed by a review of the requirements, both functional and non-functional, in section 5.4.

5.2 Testing

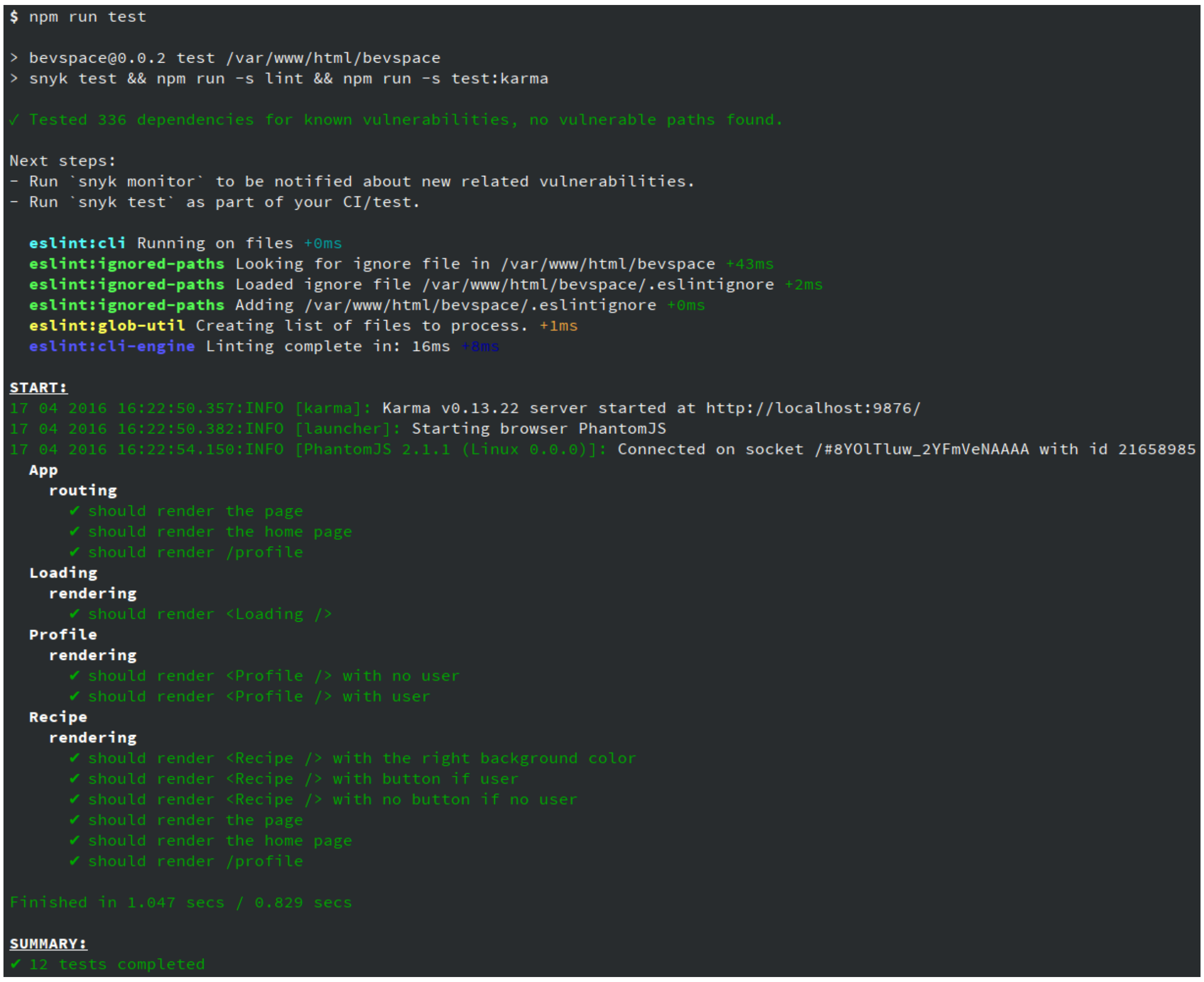
The testing framework used was mocha and complemented by the ‘chai’ and ‘sinon’ packages. This meant simple and targeted tests could be created. (*Mocha - the fun, simple, flexible JavaScript test framework*, n.d.)

Using a headless browser such as PhantomJS means tests can be run automatically in a browser environment without having to open a browser such as Chrome or FireFox. (*PhantomJS | PhantomJS*, n.d.)

To run the tests created with mocha in PhantomJS a test runner called Karma was used. (*Karma - Spectacular Test Runner for Javascript*, n.d.) This also handled processing the files through the same Webpack config that is used to build the site normally.

The tests created are for testing basic rendering of the Preact components and also that certain elements do not render if not authenticated. Figure 5.1 shows tests running in a terminal environment. [1]

[1]Note if ESLint passes linting, it usually doesn’t output anything but for this screenshot the ‘debug’ flag has been added



```
$ npm run test

> bevspace@0.0.2 test /var/www/html/bevspace
> snyk test && npm run -s lint && npm run -s test:karma

✓ Tested 336 dependencies for known vulnerabilities, no vulnerable paths found.

Next steps:
- Run `snyk monitor` to be notified about new related vulnerabilities.
- Run `snyk test` as part of your CI/test.

  eslint:cli Running on files +0ms
  eslint:ignored-paths Looking for ignore file in /var/www/html/bevspace +43ms
  eslint:ignored-paths Loaded ignore file /var/www/html/bevspace/.eslintignore +2ms
  eslint:ignored-paths Adding /var/www/html/bevspace/.eslintignore +0ms
  eslint:glob-util Creating list of files to process. +1ms
  eslint:cli-engine Linting complete in: 16ms +8ms

START:
17 04 2016 16:22:50.357:INFO [karma]: Karma v0.13.22 server started at http://localhost:9876/
17 04 2016 16:22:50.382:INFO [launcher]: Starting browser PhantomJS
17 04 2016 16:22:54.150:INFO [PhantomJS 2.1.1 (Linux 0.0.0)]: Connected on socket /#8YOlTluw_2YFmVeNAAAA with id 21658985
  App
    routing
      ✔ should render the page
      ✔ should render the home page
      ✔ should render /profile
  Loading
    rendering
      ✔ should render <Loading />
  Profile
    rendering
      ✔ should render <Profile /> with no user
      ✔ should render <Profile /> with user
  Recipe
    rendering
      ✔ should render <Recipe /> with the right background color
      ✔ should render <Recipe /> with button if user
      ✔ should render <Recipe /> with no button if no user
      ✔ should render the page
      ✔ should render the home page
      ✔ should render /profile

Finished in 1.047 secs / 0.829 secs

SUMMARY:
✔ 12 tests completed
```

Figure 5.1: Output to the terminal when running 'npm run test'

For the continuous integration, the tests running include Snyk, linting and the previously mentioned rendering tests. 3.3.5.1

5.3 Evaluation

This testing in the previous section 5.2 shows stability in the application and the foundations for a maintainable testing setup. This section will evaluate the implementation.

The project since the PID (project initiation document, see appendix A) has changed dramatically. Due to the initial client moving away from the project a change in focus occurred from from home-brewing to performance.

Overall, where the project is weakened is through the planning and structuring. With more discipline this project could have fulfilled more of the initial requirements. Using a traditional project management approach such as using Gantt charts could have helped here. Trello was not used in favour of GitHub issues. With a larger team Trello and the kanban structure would have been useful for higher-level planning.

Saying this, a lot of the technologies used in the project are in their infancy, as seen in section 4.5. This means potentially the work flow, tools and technologies will all be outdated soon and replaced with better iterations.

People and their needs don't change so much. Performance will always be important. So the research and work done in this area was very beneficial. Principles such as offline first powered by service worker caching and clientside databases as implemented in this artefact look to be the future of the web industry.

While there is testing for the application itself, user testing should have been done in order to evaluate the research reviewing in section 2.3.2. This could be included in any future work on the artefact.

5.4 Requirements Review

This section reviews the requirements set in the artefact design (see section 3.2), each requirement is reintroduced with discussion as to how successful each requirement was met.

5.4.1 Functional Requirements

This section will review the functional requirements.

5.4.1.1 Finding recipes

- As a **user**, I want to **browse for recipes**

  This is fulfilled with a list of recipes (as seen in Figure 4.2) simply showing the name of the brew, the author fo the brew and the colour of the brew as a strip on the left for context. There is also the IBU (International Bittering Unit) and ABV (Alcohol by Volume) displayed to the right.

  One problem found, when rendering close to 1000 recipes into the DOM there is significant lag. Looking into the cause of this jank it was found that there was a large amount of garbage collection. Using pagination or a virtual list would solve this problem.

5.4.1.2 Starting a brew

- As a **user**, I want to **select and start a brew from any recipe**

  When selecting a recipe there is a button to start the brew. This adds the recipe to the brews section but doesn't start any timers. This allows for the brews to be in 'staging' area. This functionality could be clearer as currently seen as starting the brew.

5.4.1.3 Managing a brew

- As a **user**, I want to **be shown what point I am in a brew**

  When a brew has started (as shown in Figure 4.2) there is a progress circle with a percentage and completed tasks in the brewing timeline are coloured green to indicate they have been completed.

  For future work this could be extended to have a 'current' task which knows how long a task should take for better context. Also push notifications could be implemented for each task another great use for a service worker. Unfortunately due to needing to use Google Cloud Messaging a Node.js server would be needed to be handle this, complicating the application. (*Cloud Messaging | Google Developers*, n.d.)

- As a **user**, I want to **add notes to key stages of a brew**

  This functionality wasn't fulfilled in the current implementation, but with CouchDB (and similarly PouchDB) not needing schemas, arbitrary data can be added to documents. Therefore this functionality, with more time would be easily implemented.

- As a **user**, I want to **review and evaluate the brew on completion**

  As with the previous functionality this wasn't fulfilled and would be fairly simple to implement.

5.4.2 Non-Functional Requirements

This section will review the non-functional requirements.

- **Compliant to WCAG 2 level AAA to ensure enough contrast "when viewed by someone having color deficits or when viewed on a black and white screen". (*Techniques For Accessibility Evaluation And Repair Tools*, n.d.)**

  When looking at the example of the items on the page, there was some need to improve the contrast.

  Fortunately using SASS (as chosen in section 3.3.4.3) this is achieved very easily by changing variables. For example, `color: \$brown;` producing `\#943938` to `color: darken(\$brown, 10\%);` producing `\#6F2B2A`.

  Table 5.1 shows how the new darkened version of the brown colour is WCAG 2 AAA compliant.

Colour	Contrast Ratio	WCAG 2 AA Compliant	WCAG 2 AAA Compliant
#943938	6.24	Yes	No
#6F2B2A	8.81	Yes	Yes

Table 5.1: Showing the compliance of the different colours against the background

- **Be documented so that others can maintain**

  For future maintaining of the project documentation should be used. The README file shows instructions to build and work with the project. Comments throughout components give explanation for code that isn't straight forward.

  A package called commitizen was used to enforce the contributing guidelines for project. (*commitizen*, n.d.)

- **Conform to the performance budget that is set in initial development**

  From the budget set in section 3.1, here are the results with the implemented site show in table 5.2.

  Notice how times change before and after caching with the service worker.

Site	**Start Render**	**Document Complete**	**Fully Loaded**
Malt.to	2.190s	3.998s	4.145s
Proposed solution	1.752s	3.198s	3.316s
Implementation	1.086s	2.118s	3.914s
Implementation w/ SW	0.890s	1.784s	4.895s

Table 5.2: Performance budget calculation with results

  As shown the performance budget at the current state of the implementation was succeeded. This would be tracked for future work.

- **Pass tests generated throughout and running through continuous integration (such as Travis CI)**

  As described in detail in section 5.2, the artefact is passing all tests including Snyk, Linting and Karma. However, by the nature of Snyk looking for vulnerabilities constantly without maintenance these tests could fail.

5.5 Testing and Evaluation Summary

This chapter introduced the testing and some evaluation of the implementation. In section 5.2 testing locally and with Travis CI is explained. This was followed by an evaluation of the implementation in section 5.4.

This page is intentionally left blank.

Chapter 6

Conclusion

6.1 Introduction

In this chapter, the project is summarised overall. Then some conclusions are drawn about the different parts of the work in section 6.2 and finally in section 6.4 future work is discussed.

6.2 Summary

Chapter 1 introduced the project defining project goals and the motivation for the project. It also gave an outline for how the report was structured.

Chapter 2 reviewed user experience through performance and an overview of home-brewing. Some background and software surrounding home-brewing was shown. Techniques, principles and examples of performance were then explained.

Chapter 3 described the proposed solution design. Tooling, methodologists, choice of technologies and data were discussed in this chapter. This is also where the requirements were made.

Chapter 4 described the system implementation of the proposed solution and testing. In this chapter, there was discussion interesting problems and the implementation overall.

Chapter 5 described the testing and evaluation. This section explained the set-up of testing, the results found from testing as well as evaluation of the project as a whole.

6.3 Conclusions

The aim of this project was to create progressive web application. The project made great introductions to many technologies to achieve this goal such as service workers for caching, offline features through synced clientside databases and performance in general.

A lot was learnt about authoring a application in this area and this project gave a starting point to many avenues of performance related development and discussion.

Overall it was found that developing with this feature set is challenging. Building a progressive web application in the current web environment is interesting with definite room to make impact in how technologies are shaped.

With technologies such as service worker being pushed by engineers rather than standard bodies there is a lot of progress, however there is a definite need for my simpler developer ergonomic implementations of service worker related features.

6.4 Future Work

Initially the requirements not fulfilled in the implementation would be the first to be considered as future work. Bringing in more complex functionality as seen in other software such as recipe editing, social network features could be explored too.

Moving away from the core concept of this project looking at integration with brewing set-ups and allowing for automation would be a great direction to take. Especially looking at projects such as the Physical Web which compliments progressive web applications. (*The Physical Web*, n.d.)

6.5 Closing words

The web is a place of ever changing challenge. For the forthcoming future, progressive web applications show a great promise to bring more accessibility to people everywhere.

References

About - ESLint - pluggable JavaScript linter. (n.d.). Retrieved 2016-04-14, from http://eslint.org/docs/about/

Aerotwist - FLIP your animations. (n.d.). Retrieved 2016-04-04, from https://aerotwist.com/blog/flip-your-animations/

Aerotwist - the cost of frameworks. (n.d.). Retrieved 2016-04-14, from https://aerotwist.com/blog/the-cost-of-frameworks/

Animations and performance | web fundamentals - google developers. (n.d.). Retrieved 2016-04-14, from https://developers.google.com/web/fundamentals/design-and-ui/animations/animations-and-performance?hl=en

Apache CouchDB. (n.d.). Retrieved 2016-04-11, from https://couchdb.apache.org/

Application cache is a douchebag · an a list apart article. (n.d.). Retrieved 2016-04-16, from http://alistapart.com/article/application-cache-is-a-douchebag

Babel · the compiler for writing next generation JavaScript. (n.d.). Retrieved 2016-04-14, from https://babeljs.io/

BeerJSON - BeerJSON. (n.d.). Retrieved 2016-04-04, from http://www.beerjson.com/

BeerXML recipe standard. (n.d.). Retrieved 2016-04-04, from http://www.beerxml.com/

Bitbucket. (n.d.). Retrieved 2016-04-11, from https://bitbucket.org/

Bower. (n.d.). Retrieved 2016-04-11, from http://bower.io/

Brave software | building a better web. (n.d.). Retrieved 2016-04-04, from https://brave.com/

BrewPi.com - the raspberry pi brewing controller. (n.d.). Retrieved 2016-04-04, from http://www.brewpi.com/

Browserify. (n.d.). Retrieved 2016-04-11, from http://browserify.org/

Cabras, I. (n.d.). *SIBA member's survey 2016.* Retrieved 2016-04-13, from http://toolbox.siba.co.uk/documents/Facts%20and%20Figures/SIBAMembersSurvey2016.pdf

Climate recipes. (n.d.). Retrieved 2016-03-29, from http://openag.media.mit.edu/climate-recipes/

Cloud messaging | google developers. (n.d.). Retrieved 2016-04-17, from https://developers.google.com/cloud-messaging/

Code, test, and deploy together with GitLab open source git repo management software | GitLab. (n.d.). Retrieved 2016-04-11, from https://about.gitlab.com/

commitizen. (n.d.). Retrieved 2016-04-11, from https://www.npmjs.com/package/commitizen

Continuous integration | ThoughtWorks. (n.d.). Retrieved 2016-04-11, from https://www.thoughtworks.com/continuous-integration

CSS triggers. (n.d.). Retrieved 2016-04-14, from https://csstriggers.com/

developit/preact. (n.d.). Retrieved 2016-02-18, from https://github.com/developit/preact

Dominant colors for lazy-loading images | manu.ninja. (n.d.). Retrieved 2016-03-29, from https://manu.ninja/dominant-colors-for-lazy-loading-images

Dont repeat yourself. (n.d.). Retrieved 2016-04-11, from http://c2.com/cgi/wiki?DontRepeatYourself

Ericsson mobility report. (n.d.). Retrieved 2016-04-13, from http://www.ericsson.com/res/docs/2016/mobility-report/ericsson-mobility-report-feb-2016-interim.pdf

Express - node.js web application framework. (n.d.). Retrieved 2016-04-14, from http://expressjs.com/

Flipkart case study. (n.d.). Retrieved 2016-03-29, from https://developers.google.com/web/showcase/case-study/pdfs/flipkart.pdf

Font loading revisited with font events | filament group, inc., boston, MA. (n.d.). Retrieved 2016-04-13, from https://www.filamentgroup.com/lab/font-events.html

Fowler, M. (n.d.). *Agile guide.* Retrieved 2016-04-04, from http://martinfowler.com/agile.html

Git - about version control. (n.d.). Retrieved 2016-04-11, from https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

GitHub. (n.d.). Retrieved 2016-04-04, from https://github.com/

Github LaTeX CI. (n.d.). Retrieved 2016-04-11, from https://www.sharelatex.com/github/

Google Chrome Developers. (n.d.). *Intro to AMP (accelerated mobile pages).* Retrieved 2016-02-12, from https://www.youtube.com/watch?v=lBTCB7yLs8Y

Google Developers. (n.d.). *Supercharging page load (100 days of google dev).* Retrieved 2016-02-29, from https://www.youtube.com/watch?v=d5_6yHixpsQ&feature=youtu.be&t=377

High performance animation - html5 rocks. (n.d.). Retrieved 2016-04-14, from http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

How to homebrew beer. (n.d.). Retrieved 2016-04-16, from https://byo.com/newbrew

How we make RWD sites load fast as heck | filament group, inc., boston, MA. (n.d.). Retrieved 2016-04-13, from https://www.filamentgroup.com/lab/performance-rwd.html

Index | greenkeeper. (n.d.). Retrieved 2016-04-17, from https://greenkeeper.io/

Introducing RAIL: A user-centric model for performance. (n.d.). Retrieved 2016-01-18, from https://www.smashingmagazine.com/2015/10/rail-user-centric-model-performance/

Jehl, S., Parker, T., & Bender, J. (n.d.). *Performance impact of popular JavaScript MVC* *frameworks.*

JSX in depth | react. (n.d.). Retrieved 2016-04-14, from `https://facebook.github.io/react/docs/jsx-in-depth.html`

Karma - spectacular test runner for javascript. (n.d.). Retrieved 2016-04-17, from `https://karma-runner.github.io/0.13/index.html`

LaTeX – a document preparation system. (n.d.). Retrieved 2016-04-11, from `https://www.latex-project.org/`

Let's encrypt - free SSL/TLS certificates. (n.d.). Retrieved 2016-04-17, from `https://letsencrypt.org/`

LukeW | mobile design details: Performing actions optimistically. (n.d.). Retrieved 2016-04-04, from `http://www.lukew.com/ff/entry.asp?1759`

Maceri, K. (n.d.). *Document design for user with reading disorders.* Retrieved from `http://www.angelfire.com/tn3/writing/DesignUsersReadDis.pdf`

Mall, D. (n.d.). *How to make a performance budget.* Retrieved 2016-02-18, from `http://danielmall.com/articles/how-to-make-a-performance-budget/`

Malt.io blog: Introducing brauhaus.js. (n.d.). Retrieved 2016-04-11, from `http://blog.malt.io/2013/03/introducing-brauhausjs.html`

Mocha - the fun, simple, flexible JavaScript test framework. (n.d.). Retrieved 2016-04-17, from `https://mochajs.org/`

Moment.js | home. (n.d.). Retrieved 2016-04-11, from `http://momentjs.com/`

Node.js. (n.d.). Retrieved 2016-04-14, from `https://nodejs.org/en/`

npm. (n.d.). Retrieved 2016-04-14, from `https://www.npmjs.com/`

Passport. (n.d.). Retrieved 2016-04-14, from `http://passportjs.org/`

Performance is user experience | designing for performance. (n.d.). Retrieved 2016-04-13, from `http://designingforperformance.com/performance-is-ux/`

PhantomJS | PhantomJS. (n.d.). Retrieved 2016-04-17, from `http://phantomjs.org/`

The physical web. (n.d.). Retrieved 2016-04-17, from `https://google.github.io/physical-web/`

postcss/autoprefixer: Parse CSS and add vendor prefixes to rules by can i use. (n.d.). Retrieved 2016-04-14, from `https://github.com/postcss/autoprefixer`

postcss/postcss: Transforming styles with JS plugins. (n.d.). Retrieved 2016-04-14, from `https://github.com/postcss/postcss`

Progressive enhancement: Zed's dead, baby : Tom dale. (n.d.). Retrieved 2016-04-14, from `http://tomdale.net/2013/09/progressive-enhancement-is-dead/`

Progressive enhancement — government service design manual. (n.d.). Retrieved 2016-04-14, from `https://www.gov.uk/service-manual/making-software/progressive-enhancement.html`

Progressive web apps | web fundamentals - google developers. (n.d.). Retrieved 2016-04-18, from https://developers.google.com/web/progressive-web-apps?hl=en

Rendering performance | web fundamentals - google developers. (n.d.). Retrieved 2016-04-16, from https://developers.google.com/web/fundamentals/performance/rendering/?hl=en

Research blog: Speed matters. (n.d.). Retrieved 2016-04-04, from http://googleresearch.blogspot.co.uk/2009/06/speed-matters.html

Research briefings - pub companies, pub tenants & pub closures: background history (up to 2014). (n.d.). Retrieved 2016-04-13, from http://researchbriefings.parliament.uk/ResearchBriefing/Summary/SN06740

rollup/rollup. (n.d.). Retrieved 2016-02-20, from https://github.com/rollup/rollup

Sass: Syntactically awesome style sheets. (n.d.). Retrieved 2016-04-14, from http://sass-lang.com/

Say hello to offline first. (n.d.). Retrieved 2016-04-04, from http://hood.ie/blog/say-hello-to-offline-first.html

Service worker API - web APIs | MDN. (n.d.). Retrieved 2016-04-04, from https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

Service workers replacing AppCache: a sledgehammer to crack a nut — medium. (n.d.). Retrieved 2016-04-17, from https://medium.com/@firt/service-workers-replacing-appcache-a-sledgehammer-to-crack-a-nut-5db6f473cc9b#.1pewk1mem

Setting a performance budget - TimKadlec.com. (n.d.). Retrieved 2016-04-13, from https://timkadlec.com/2013/01/setting-a-performance-budget/

Snyk. (n.d.). Retrieved 2016-04-14, from https://snyk.io/

Speed index - WebPagetest documentation. (n.d.). Retrieved 2016-04-13, from https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index

Storey, D. (n.d.). *@helloanselm you wouldn't really get the benefit of will-change in edge. it is already optimised in a way that it is fast.* [microblog]. Retrieved 2016-04-14, from https://twitter.com/dstorey/status/707990118485721088

Surge. (n.d.). Retrieved 2016-04-11, from https://surge.sh/

Taylor, D. (n.d.). *Malt.io Blog: Malt.io is Born.* Retrieved 2016-01-18, from http://blog.malt.io/2012/09/maltio-is-born.html

Techniques for accessibility evaluation and repair tools. (n.d.). Retrieved 2016-04-15, from https://www.w3.org/TR/AERT#color-contrast

Trello. (n.d.). Retrieved 2016-04-04, from https://trello.com

Usability engineering (1edition ed.). (n.d.). Morgan Kaufmann.

Using ServiceWorker in chrome today - JakeArchibald.com. (n.d.). Retrieved 2016-04-15, from https://jakearchibald.com/2014/using-serviceworker-today/

Using web workers - web APIs | MDN. (n.d.). Retrieved 2016-04-04, from https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/

Using_web_workers#Other_types_of_worker

Webfonts - writing. (n.d.). Retrieved 2016-03-18, from http://mrmrs.io/writing/2016/03/17/webfonts/

webpack module bundler. (n.d.). Retrieved 2016-04-11, from https://webpack.github.io/

What is user experience design? overview, tools and resources – smashing magazine. (n.d.). Retrieved 2016-04-13, from https://www.smashingmagazine.com/2010/10/what-is-user-experience-design-overview-tools-and-resources/

Why JSON will continue to push XML out of the picture - CenturyLink cloud developer center. (n.d.). Retrieved 2016-04-16, from https://www.ctl.io/developers/blog/post/why-json-will-continue-to-push-xml-out-of-the-picture

Why web performance matters: Is your site driving customers away? (2011, October). Retrieved from http://www.mcrinc.com/Documents/Newsletters/201110_why_web_performance_matters.pdf

This page is intentionally left blank.

Appendices

Appendix A

Project Initiation Document

A.1 Basic Details

Student Name	Zac Colley
Draft project title	Free Automated Interconnected Beverage Brewing
Course	Web Technologies
Client organisation	GetBrewing.uk
Client contact name	Alan Thompson
Project supervisor	Rich Boakes

A.2 Outline of the project environment and problem to be solved

In this project I aim to develop a web-based system that will support home-brewers and help to ensure a successful and repeatable brewing process. The system may integrate with brewing hardware and web services in order to provide record and act upon real-time brewing data.

The project is in partnership with Alan Thompson from GetBrewing.uk, a home brewing company on Elm Grove in Portsmouth.

One of the his products, the Grainfather as a simple control and sensor system. It currently has: a temperature sensor with an LED screen to display this temperature, manual control of the pump and manual control of the heating element.

He wishes to offer his brewing kit with an improved control system.

The proposed solution would improve the Grainfather and help record, collect, share and analyse data amongst brewers.

Apart from the commercial aspect of the project, there is other scientific exploration.

With so many varied set-ups for home brewing, recipes can be tweaked and processes can be changed. The project will explore a web interface to complement the current home-brewing tech eco-system.

A.3 Project aim and objectives

A.3.1 Aims

- Improve home-brewing processes through design and development of a web application.
- There will be a strong focus on user experience research and use of modern development practises.

A.3.2 Objectives

- Research and apply an designed solution towards a better user experience
- Based on UX research, create an intelligent web interface connected to the brewing kit for:
  - Guiding and instructing brewer on current brew
  - Recording notes and actions throughout the brew process
  - Connecting to current brewing software seamlessly
- Analyse user experience approach through structured testing and evaluate the results.

A.4 Project deliverables

Potential deliverables include:

- Web interface for a brewing process versioning system
- Project report
- PID
- Trello board

A.5 Project constraints

Some of the user interface work would benefit from working on a real-life working brewing kit. While Alan is a contributor to the project, his equipment, specifically the brewing kit is not always accessible for all of the design and development of the web interface. This will mean assumptions must be made and some simulations of the running environment where needed.

A.6 Project approach

I am lacking a lot of knowledge and skills in brewing, this will take some learning. I have contacted other members in the University who have research interests in brewing to learn more.

I am approaching the development with a sprint based model in mind. Simply prioritising which work needs to be tackled in each sprint to reach the goal at hand. This goal may be more research driven or more technically driven depending on the stage of work.

A.7 Facilities and resources

No specialist equipment will be needed from the School of Computing as most of the work can be done on standard PC set-ups.

Research materials from the library will be useful for research.

For the web interface due to the nature of the application of the software it will require real-time technologicals such as an event-driven web scale server (Node.Js) and websockets (through Socket.IO).

Most of the software used will be free to use and so will not require funding. However any of the hardware to interface with the brewing kit would have to be purchased.

A.8 Log of risks

The main risk is the project requires a large amount of hours so any other concerns such as contract work or other course work will lessen the amount of hours.

A.9 Starting point for research

Through talking to the client more areas of research will be made available.

In terms of the hardware two good places to look are the Open ArdBir and BrewPi projects, both specifically for home brewing.

Finding resources based on user experience in similar areas (machinery, cooking) will make sense.

A.10 Breakdown of tasks

This my initial tasks to start the project:

- Research home brewing
  - Review previous materials

    - Talk to Dr Jeremy Mills

- Research and choose the technology stack
    - For backend versioning system
        - Graph databases?
        - CouchDB
        - Git
    - For the fronted web interface
        - React
        - Node.js (Express, Socket.IO)

A.11 Project plan

I will be using Trello to organise my tasks, at each sprint I will work with my supervisor and Alan (where needed) to work out what needs to be put in the sprint. There will also be a backlog of tasks (such as research, development) which will allow a overview of the project easily. This naturally will give an idea of future work for the project too.

A.12 Legal, ethical, professional and social issues

There are no initial pressing legal/ethical/professional/social issues that may impose constraints on the project.

The project has been approved on the ethics website under the same name.

Appendix B

Ethics Certificate

University of Portsmouth

Certificate of Ethics Review

Project Title:	Free Automated Interconnected Beverage Brewing
User ID:	665219
Name:	Zac Andrew Colley
Application Date:	10/10/2015 16:22:08

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative for the School of Computing is Carl Adams

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

SchoolOrDepartment: SOC
PrimaryRole: UndergraduateStudent
SupervisorName: Rich Boakes
HumanParticipants: No
PhysicalEcologicalDamage: No
HistoricalOrCulturalDamage: No
HarmToAnimal: No
HarmfulToThirdParties: No
OutputsPotentiallyAdaptedAndMisused: No
Confirmation-ConsideredDataUse: Confirmed
Confirmation-ConsideredImpactAndMitigationOfPontentialMisuse: Confirmed
Confirmation-ActingEthicallyAndHonestly: Confirmed

Certificate Code: F6DB-9C78-4CDB-335A-BCA3-F7B7-236B-EAC0 Page 1

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor signature:

Date:

Figure B.1: The author well slept with bananas