Technical report: progress with developing an outcome-based web application

RESAS1.2.4.3 [D6 Technical report]

Authors: Kit (C.J.A.) Macleod* and Richard Hewitt, James Hutton Institute, Aberdeen, UK.

*Corresponding author: kit.macleod@hutton.ac.uk

Suggested citation: Macleod, C.J.A. and R. Hewitt (2018). Technical report: progress with developing an outcome-based web application. James Hutton Institute.

Contents

Executive summary

This technical report sets out progress with developing a spatially explicit Facilitated Outcome-based Land Management (FOLM) web application based on user stories developed from functional and non-functional requirements provided by stakeholders. We present: a summary of previous development activities; an overview of developing mobile web and native mobile applications, including general points to be aware of when developing mobile web applications; and examples of development options related to how we implement draft user stories in our web application.

There are six main linked challenges to producing useful working web applications: 1) understanding what people actually need from your software- to aid an existing or new task, 2) deciding exactly what you will build i.e. a requirements process, 3) deciding how you will implement those requirements, 4) building a working web application, 5) testing your software to ensure that it works as expected, and 6) an overarching challenge related to common development practice of considering and dealing with the first five challenges in an iterative and continuous manner.

In this report we focus on the third of these challenges- deciding how to implement stakeholder prioritised requirements, and its connections to the second (deciding exactly what you will build), fourth (building a working web application) and sixth challenges (iterative and continuous development).

The first two phases of our development process focussed on the first two challenges. From an initial broad review of software options, we learned that software options grouped under the heading 'software packages and applications for developing web-based applications' were most likely to meet our initial screening criteria. Reponses from 14 interviews highlighted the need for practical tools to facilitate decision making about land and water management based on a range of environmental and financial outcomes. We developed and demonstrated an initial spatially explicit web-based prototype during a stakeholder workshop. Feedback from this demonstration was grouped under five broad headings: 1) supporting land manager decisions, 2) an easy to use and adaptive tool/application, 3) evidence and uncertainty, 4) the scale at which the tool/application operates, and 5) interactions between actions and their impact on environmental state.

In this report we summarise four broad approaches to building web and mobile applications. To meet the requirement for our application to work online and offline across a range of platforms, we decided to implement a progressive web application.

Then we highlight three general points to be aware of when developing web applications: the first is that a web application is dependent on a wide range of technologies; second, there is no single solution to meeting the requirements, as there are several groups of technologies (often referred to as software/solution stacks) that can be used for creating web applications; and the third point is these technologies and associated best practices are constantly evolving.

There are a range of options when deciding to implement map related functionality in our application. We provide a summary of the most relevant options that have potential to meet our requirements e.g. free technologies that enable dynamic maps in web applications. These are OpenLayers, Leaflet, and Mapbox GL. Our next steps are to continue working on challenges four (building a working web application), five (testing your software to ensure that it works as expected) and six (iterative and continuous development).

1. Introduction

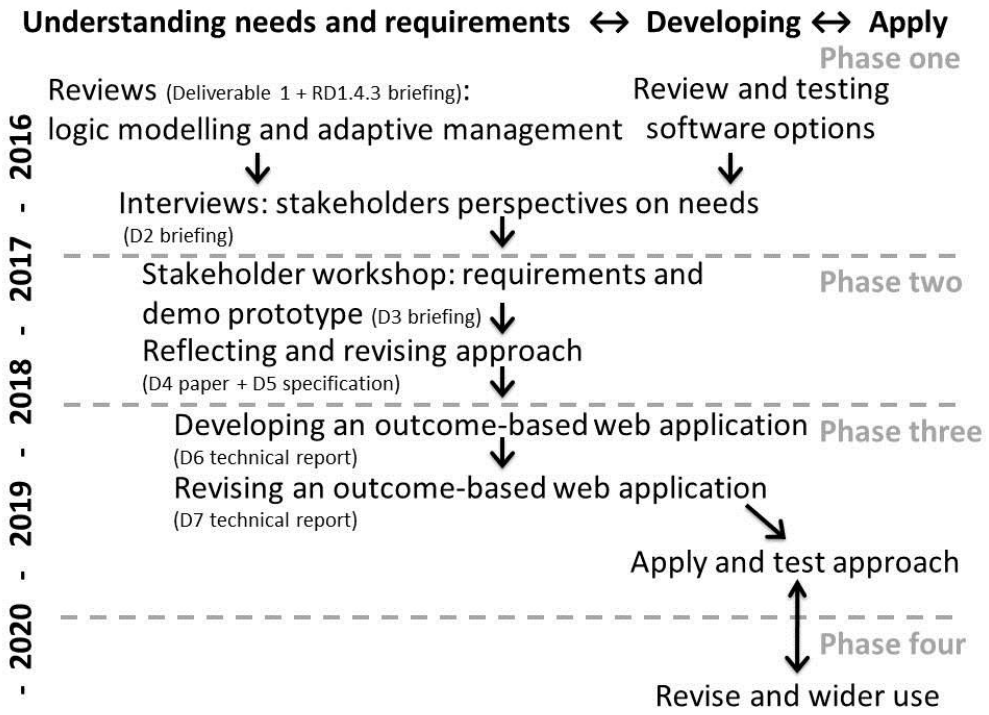
1.1 Purpose

This technical report sets out progress with developing an outcome-based web application through user stories[1] created from stakeholder prioritised functional[2] and non-functional requirements[3] (see Figure 1 and (Macleod and Hewitt, 2018)). Where user stories are written from the perspective of a person using the software, and are short, high-level descriptions of functionality i.e. high-level definition of what the software is capable of doing. In this document we present a summary of previous development steps, an overview of developing mobile web and native mobile applications[4] including general points to be aware of when developing mobile web applications, and examples of development options related to how we implement requirements of the outcome-based web application (see Figure 1 and (Macleod and Hewitt, 2018)).

1.2 Background

This technical report (D3.6) is part of the Scottish Government Strategic Research Programme (SRP) project 'Assessment of the effectiveness of interventions to achieve water policy objectives (RESAS 1.2.4 Objective 3)' developing a Facilitated Outcome-based Land Management (FOLM) application to aid land manager decision-making for multiple benefits (Figure 1). There is increasing interest in how we improve targeting of land and water management actions e.g. Scottish Rural Development Programme[5] (SRDP) management options for one or more environmental outcomes.

Figure 1 Outline of our process to develop an outcomes-based approach



[1] https://en.wikipedia.org/wiki/User_story
[2] https://en.wikipedia.org/wiki/Functional_requirement
[3] https://en.wikipedia.org/wiki/Non-functional_requirement
[4] https://www.pcmag.com/encyclopedia/term/47651/native-application
[5] http://www.gov.scot/Topics/farmingrural/SRDP

1.3 Overview of six linked challenges related to producing web applications

Five of the six main linked challenges to producing useful working web applications are: 1) understanding what people actually need from your software- to aid an existing or new task, 2) deciding exactly what you will build i.e. a requirements process, 3) deciding how you will implement those requirements, 4) building a working web application, and then 5) testing your software to ensure that it works as expected. In earlier reports we explored what people may need from our outcome-based web application (Macleod and Hewitt, 2017b, Macleod and Hewitt, 2017a, Macleod and Hewitt, 2018); Figure 1). In a related report (Figure 1, D5) we present a list of prioritised requirements presented as draft user stories- to set out specific functionality or constraints the software needs to satisfy (Macleod and Hewitt, 2018). In Vitolo *et al*. (2015), the lead author discussed the third and fourth challenges of deciding how to implement a web application and building it, with reference to development of a flooding application and what was learned. The practice of software development is littered with examples of good practice and tools to aid the testing of software e.g. (Myers et al., 2011). A sixth and overarching challenge relates to the common development practice of considering and dealing with these five linked challenges in an iterative and continuous manner.

In this report we **focus on the third of these challenges- deciding how to implement prioritised requirements,** and its connection to the second (deciding exactly what you will build), fourth (building a working web application) and sixth challenges (iterative and continuous development). As set out in the related report (Figure 1, D5), we are using best practices from people-centred development process including: analyse requirements and needs, design for usability by prototyping[6], evaluate in context, and gather feedback to aid planning the next iteration (Gulliksen et al., 2003, Macleod and Hewitt, 2018). This technical report is a stepping-stone to a later technical report (Figure 1, D7) on revising an outcome-based web application- focussing on challenges four, five and six.

2. Summary of previous application development steps

In this section we briefly summarise previous development activities in the first two phases and what we have learned (Figure 1); these activities have focussed on the first two challenges (Section 1.3).

2.1 Initial broad review of approaches and software options

After reviewing the theory and practice of logic modelling and adaptive management[7], and discussing these with stakeholders (Figure 1); we reviewed potential software options for producing our outcome-based approach, using a set of screening criteria (Table 1) (Macleod and Hewitt, 2017a, Hewitt and Macleod, 2017). **From this review we learned that software options grouped under the heading 'software packages and applications for developing web-based applications' were most likely to meet our initial screening criteria** (Table 1). Further details of this activity can be found in (Macleod and Hewitt, 2017a, Hewitt and Macleod, 2017).

[6] https://en.wikipedia.org/wiki/Prototype
[7] https://en.wikipedia.org/wiki/Adaptive_management

2.2 Understanding stakeholder needs and developing an initial set of requirements

We carried out interviews with 13 regional and national level stakeholders involved with natural resource management[8] in Scotland. Interviewees were asked about their needs for developing a more integrated approach to land use and catchment management using incentives and regulations for the delivery of multiple benefits. **Their responses highlighted the need for practical tools to facilitate decision making about land and water management based on a range of environmental and financial outcomes** (Macleod and Hewitt, 2017a).

We then arranged a workshop, and asked a group of participants to rate 17 'needs' which had been extracted from the earlier interviews. A need was defined as something we wanted to address with the approach and software application. The participants were invited to "please rate how important is this need for developing tools to support decisions about the effectiveness of land management interventions for multiple benefits?" Details of the workshop can be found in (Macleod and Hewitt, 2017b).

2.3 Development and testing of a spatially explicit web-based prototype

Figure 2 Touch table demonstration of spatially explicit web-based prototype

2.3.2 Design of prototype based on seven screening criteria

The design of the prototype was undertaken in-line with seven screening criteria that had emerged through consultations with stakeholders (Figure 1; (Hewitt and Macleod, 2017)):

[8] https://en.wikipedia.org/wiki/Natural_resource_management

1. *Should be free at the point of use*

Development was undertaken using R[9], a well-known free and open-source computing environment for statistical computing and graphics. The start-up script was written in Visual Basic Scripting language (VBScript)[10], which is available to all Windows users. The start-up script was optional, and the prototype can be used without it.

2. *Should work on touch devices like mobile phones, tablets and larger touch tables*

The prototype was developed for, and tested on a large touch table running Windows 10; to enable demonstration and discussion with a small group.

3. *Should have map-based functionality for users to interact with spatial information e.g., information on fields and other features related to land and water management*

The prototype comprised a large map window with interactive button functionality on the right hand side to allow people to interact with the map. The map-based functionality was provided by the open source Leaflet[11] JavaScript library, which is designed to provide mobile friendly interactive maps (Figure 3).

The prototype included data sets on hydrological sub-catchments and land cover in ESRI shape file format[12] (the best-known and most widely used GIS vector data format), and was explicitly targeted at land and water management stakeholders. The free and open data source OpenTopoMap[13] was chosen as a background/base map since it is attractive, easily available and provides topographic information, but any other map layer (e.g. open OS data[14]) could be used.

Interactions included: selecting a sub-catchment from a drop-down list box (and causing the map to automatically zoom to the chosen sub-catchment); identifying map features with a single touch; and extracting a portion of the underlying land cover layer (LCM2007[15]) by drawing a polygon on the screen (using the Leaflet draw plugin[16]), and automatically updating the attribute table of the clipped out polygon with its calculated area; and visualisation of the polygon's range of land cover areas as a bar chart (Figure 3).

4. *Should include functionality for outcomes-based logic models i.e. linking land management to a range of outcomes*

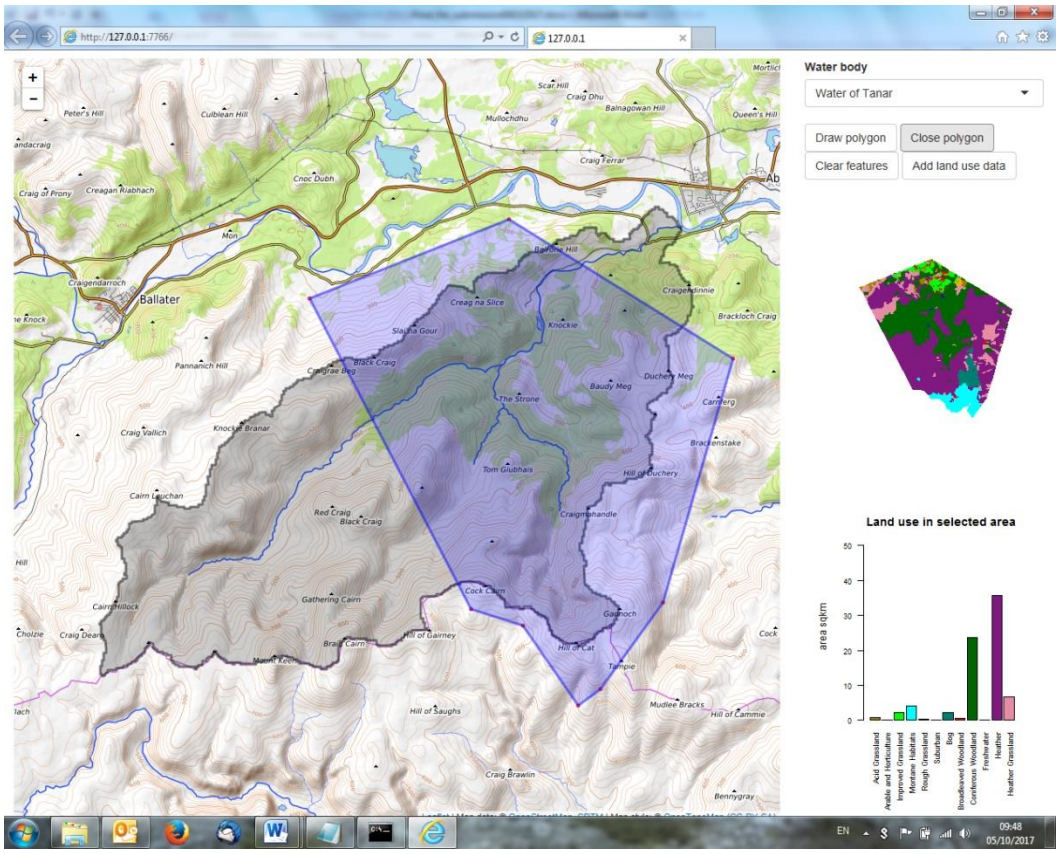
This functionality was not fully implemented at this stage. The functionality in the above section (3) is a prerequisite to this more advanced functionality.

5. *Should allow developers and end-users to develop and extend the software/existing application*

The prototype could be extended by developers according to user requirements; it was not possible for users to modify the prototype directly.

[9] https://www.r-project.org/
[10] https://en.wikipedia.org/wiki/VBScript
[11] http://leafletjs.com/
[12] https://en.wikipedia.org/wiki/Shapefile
[13] https://opentopomap.org/#map=5/49.023/10.063
[14] https://www.ordnancesurvey.co.uk/business-and-government/products/opendata.html
[15] https://www.ceh.ac.uk/services/land-cover-map-2007
[16] http://leaflet.github.io/Leaflet.draw/docs/leaflet-draw-latest.html

Figure 3 Prototype application running on Windows



Note: This screen capture shows the Water of Tanar sub-catchment of the river Dee in Aberdeenshire, Scotland. The user has drawn a polygon, which has been clipped out the land cover base map (LCM2007) and this is displayed to the right of the map window; below it, the area of each land cover in the polygon is displayed.

6. Should have potential for scientific innovation

Visualisation of complex data in a manageable way is an important area of development in several areas. The prototype itself was not innovative in a technological sense, since a range of more advanced interactive web applications for visualisation and manipulation of map-based data exist under the emerging free and open source web visualisation dashboard paradigm e.g. Tableu's Public[17], and Plotly's Dash[18]. Yet these tools are generic, rather than co-designed together with a specific user community, like our prototype, and little used in environmental management and modelling. Thus the development and practical testing of an interactive tool of this kind with a specific land manager community on the ground would be innovative.

7. Should be actively maintained, preferably through a large, open user community

The R computer language and environment is under active development, for example over the past six months a number of the procedures used in the prototype have been replaced by more advanced functionality; the user community is very large, and growing.

[17] https://public.tableau.com/en-us/s/
[18] https://plot.ly/products/dash/

2.3.2 Stakeholder feedback

Whilst demonstrating and discussing this initial prototype we received a range of feedback (Table 2). These were grouped under five broad headings: 1) supporting land manager decisions, 2) an easy to use and adaptive tool/application, 3) evidence and uncertainty, 4) the scale at which the tool/application operates, and 5) interactions between actions and their impact on environmental state (Table 2).

Table 2 Summary of the discussed needs

Supporting land manager decisions
Land management business needs to know where to invest.
Land managers want to know the "value" (not necessarily just in monetary terms)* of a project benefit.
How to provide the information best so that land manager or their agent can use it?
Objectives are consistent but mechanisms are uncertain.
Land managers need to prioritise their activities based on the value of benefit to them. They are used to making decisions under uncertainty, evidence helps reduce any risks.
We are talking about options – multiple options under diverse scenarios.
There is a trade-off between diversity (of habitats, species) and [ecological]* connectivity. Thus, while more diversity may be desirable in some respects, it may reduce connectivity.
Easy to use and adaptive tool/application
Key consideration is how easy is the tool to use? Complex technical tools may not get used.
Tool must be adaptive to political changes/policy changes e.g. Brexit.
People want detailed information, but this is place specific, so a generic tool may not help. There is a clear tension between site-specific information and general approaches.
There is in fact no single answer to "what's best". Need a system tool that is weighted towards obtaining diverse outcomes.
Evidence and uncertainty
Evidence is vital. Need to be very careful how to present it given its inherent uncertainty.
Balancing and managing risk under uncertainty. Greater local focus may mean more uncertainty in data available.
Scale at which the tool/application operates and
Key question is the scale at which the tools operate.
Three main users for a potential tool can be identified: Grant giving body– want to know how best to use public money; Land manager – wants to know what impact will this have, what options are available to me?; and Communities – need to help communities in collective decision making.
Interactions between actions, and their impact on state of the environment
Interventions can change what is needed (at later times and in other locations). There is a cumulative effect [around uptake of SRDP options] such that there may be diminishing returns, i.e. once my neighbour does it may be less valuable for me to do it. Could this be reflected somehow in tools?
*Curved parentheses indicate insertions by the speaker during their own intervention; square parentheses indicate insertions by the facilitator, either during the intervention or while compiling the report. The facilitators' insertions were attempts to clarify something that was evident.

2.3.3 What learned

The feedback in Table 2 reinforced the previous identified need for simple to use tools to aid land manager decision making about management actions and linked to evidence of potential environmental outcomes. Demonstrating the prototype was invaluable; as it highlighted that an interactive spatially explicit application could meet land manager needs. Further details from the

workshop are presented in (Macleod and Hewitt, 2017b). One important next step highlighted during the workshop, was the need to implement offline functionality; this would also be innovative, since most web-based dashboard type applications require an active internet connection.

The prototype demonstrated the ability to manipulate spatial information in the form of a land cover map and its attributes, which can be: removed, added, recalculated and displayed as desired, subject to minor modifications to the program. The polygon area was chosen as an easily understandable demonstration attribute, but clearly any data that can be associated with a polygon such as soil type can be queried in this way. Examples not implemented could include estimated nutrient export (given a nutrient export/ha approximation) or cost of some measure (given a cost/ha approximation).

3. Overview of developing mobile web and native mobile applications

The purpose of this section is to provide an overview of the state of play with developing mobile application technologies. The focus is on the development of our application and is not meant to be a comprehensive review.

3.1 Summary of four broad approaches to building mobile web and native mobile applications

When developing for mobile platforms, you need to be aware of the different hardware characteristics compared to desktop or laptop computers: for example working with small touch screens and different application programming interface[19] (APIs) e.g. geolocation.

There are (at least) four broad types of technologies for building web or mobile applications. Progressive web applications[20] are based on traditional web applications that can be viewed through a web browser e.g. Chrome or Firefox, and are built with HTML5[21] and JavaScript[22]. HTML5 is the latest version of the standard that defines HyperText Markup Language (HTML) that enables more diverse and powerful web sites and applications to be built. The reason they are called progressive web applications is that a range of technologies enables improved user experience, for example enabling faster loading of content and offline working.

A second type are native mobile applications, which you downloaded from an app stores e.g. Android[23] apps from Google Play Store that are built with Java[24] or Kotlin[25], or Apple[26] apps from the Apple App Store which are built with Objective C[27] or Swift[28]. Native mobile applications are written and optimised to work on one specific platform e.g. on Apple mobile devices. They are often fast, and once installed, work offline. However, they can be expensive to produce as you need specific

[19] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
[20] https://developers.google.com/web/progressive-web-apps/
[21] https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5
[22] https://developer.mozilla.org/bm/docs/Web/JavaScript
[23] https://developer.android.com/index.html
[24] https://java.com/en/
[25] https://kotlinlang.org/
[26] https://developer.apple.com/
[27] https://developer.apple.com/documentation/objectivec?language=objc
[28] https://developer.apple.com/documentation/swift

programming expertise to produce them, and different codebases[29] are required for multiple mobile platforms.

A third type of mobile applications are called hybrid mobile applications; these are built using specific technologies e.g. Cordova[30] or the Ionic[31] framework (based on the Angular JavaScript framework). These technologies allow you to use a single HTML/JavaScript codebase across several platforms, by wrapping then in a native container for Android or Apple mobile devices. These technologies enable fairly easy development of applications that will work on mobile devices from a single codebase (to a certain extent). Their performance and functionality is limited due to being implemented through browser-based WebViews within the native mobile platform[32].

A fourth type of technology for building mobile applications allows you to use JavaScript to create a native mobile application: a popular example is React Native[33]. Advantages of using React Native are that a 'real' mobile application can be produced using the same building blocks of iOS and Android apps, and allow you to include native code when needed. Disadvantages include the different codebases for each mobile and desktop/laptop platform.

Each of these four options for building mobile applications have relative advantages and disadvantages, and these depend on who (and what expertise they have) is building what application, under what constraints. The purpose of this section was to provide an overview of these technologies and not to present a comprehensive review.

In this report (and project) we are currently focused on building a progressive web application; as this is the simplest way to develop a map based application that can be used across a range of platforms e.g. desktop and mobile devices, offline or online.

3.2 Three general points to be aware of when developing mobile web applications

There are three general points to be aware of when developing mobile web applications (for example a progressive web app): the first is that a web application is dependent on a wide range of technologies; second, there is no single solution to meeting the requirements, as there are several groups of technologies (often referred to as software or solution stacks[34]) that can be used for creating web applications; and the third point is these technologies and associated best practices are constantly evolving.

3.2.1 Web applications are dependent on a wide range of technologies

Traditionally web applications involve a range of software technologies that communicate between desktop/laptop/mobile devices (often referred to as the client-side[35]) with other computers called servers (often called the server-side[36]). Software on servers would respond to requests from client side e.g. when data was submitted in a form or a button was pressed. The software is built using a

[29] https://en.wikipedia.org/wiki/Codebase
[30] https://cordova.apache.org/
[31] https://ionicframework.com/
[32] https://cordova.apache.org/docs/en/latest/guide/hybrid/webviews/index.html
[33] https://facebook.github.io/react-native/
[34] https://en.wikipedia.org/wiki/Solution_stack
[35] https://en.wikipedia.org/wiki/Client-side
[36] https://en.wikipedia.org/wiki/Server-side

range of computer languages e.g. on the client-side: HTML[37] describes and defines the content of a web page; along with Cascading Style Sheets (CSS)[38], a stylesheet language, that describes how HTML documents are presented; and the JavaScript programming language which is used as a scripting language for making web pages and applications interactive (it can also be used in non-browser environments). This occurs through manipulation of the Document Object Model[39] (DOM), which is an object-orientated representation of a web page[40]. More recently, a wider range of the functionality provided by server-side code can also be carried out on the client-side (see Section 4.1.1); one reason for this is that communications between the client-side and server-side can be time consuming. Nearly all web and mobile applications include additional technologies: for example they (nearly) all make use of some form of database technology to organise collections of data[41], as well as software that enables the visualisation of data in graphs and/or maps. There is far greater detail to these technologies and their use; this simplified description is meant to illustrate the diversity of technologies that are utilised to create progressive web applications, and other mobile applications.

3.2.2 A range of technology stacks can be used for creating web applications

In addition to common technologies e.g. particular database software that can be found in a wide range of dynamic web applications[42]. There are distinct technology stacks that are the main components of web applications. The archetypal stack is referred to as LAMP[43], which is an acronym of the four main technologies: the Linux[44] operating system (on the server), the Apache HTTP Server[45] (deals with requests from the browser and sends responses), the MySQL[46] relational database (on the server holds data about the application and its use), and the PHP[47] programming language (used on a server to write the application, that was interpreted by the web server software). There are many variants of these technologies.

The current range of technology stacks is far more diverse than this simple LAMP example. One reason for this diversity is the increase in use of JavaScript on the server and client side. Examples of JavaScript stacks include MEAN[48] or MERN[49]: standing for MongoDB[50] (NoSQL[51] database), Express[52] (JavaScript web server), Angular[53] or React[54] (two major JavaScript frameworks for creating web applications) and Node[55] (a JavaScript runtime that is popular in part due to its ability to handle

[37] https://developer.mozilla.org/en-US/docs/Web/HTML
[38] https://developer.mozilla.org/en-US/docs/Web/CSS
[39] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
[40] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
[41] https://en.wikipedia.org/wiki/Outline_of_databases
[42] https://en.wikipedia.org/wiki/Dynamic_web_page
[43] https://en.wikipedia.org/wiki/LAMP_(software_bundle)
[44] https://en.wikipedia.org/wiki/Linux
[45] https://httpd.apache.org/
[46] https://en.wikipedia.org/wiki/MySQL
[47] https://en.wikipedia.org/wiki/PHP
[48] http://mean.io/
[49] http://mern.io/
[50] https://www.mongodb.com/
[51] https://en.wikipedia.org/wiki/NoSQL
[52] https://expressjs.com/
[53] https://angular.io/
[54] https://reactjs.org/
[55] https://nodejs.org/en/

asynchronous events). Once again, this is a very simplified explanation of the technology stacks involved with creating web applications.

3.2.3 These technologies and associated best practices are constantly evolving

The third point is the changing nature of the technologies involved in building web applications. Some of these changes are small e.g. updates to particular software packages or libraries, but others are large e.g. increased used of modularisation with JavaScript. For example large changes to JavaScript in 2015 (officially called ECMAScript2015[56]) introduced the use of modules to enable developers to split up their code into multiple files, and encouraged code reuse. Changes to standards like this example, then produce a wave of changes to the software that uses or is dependent on JavaScript: including new technologies and best practices e.g. using module bundlers like Webpack[57].

4. Deciding how to implement requirements of the application

In a related report 'Developing an outcome-based web application: principles and requirements specification' we set out lists of functional and non-functional requirements based on user stories (Macleod and Hewitt, 2018). In this section we explore software options for achieving the highest rated and fundamental functional requirements, whilst meeting key non-functional requirements (Table 3). These initial user stories need to be unpacked, to set out what functionality is exactly required.

4.1 Deciding how to implement requirements for off line working across a range of platforms

To meet the requirements for our application to be able to work online and offline across a range of platforms, we decided to implement a progressive web application. Progressive web applications are changing how mobile and web applications are being developed[58]. These changes include: 1) improve user experience though more reliable apps (e.g. load instantly), with faster responses to user interactions (e.g. response to selecting an item or location on a map), and feel like a native Android or iOS apps, even when no network is available (e.g. 2-4G or wifi signal); and 2) enable developers and researchers to produce a single application, that can be used across all mobile and web platforms.

Developing progressive web applications involves following best practices e.g. progressive web app checklist[59] and using related technologies e.g. service workers[60]. It is essential to ensure the pages are responsive on mobile devices. Google has produced the free and open source Lighthouse tool for automated testing of progressive web applications[61].

[56] http://www.ecma-international.org/ecma-262/6.0/index.html
[57] https://webpack.js.org/
[58] https://developers.google.com/web/progressive-web-apps/
[59] https://developers.google.com/web/progressive-web-apps/checklist
[60] https://www.w3.org/TR/service-workers/
[61] https://developers.google.com/web/tools/lighthouse/

Table 3 Initial requirements to guide development of our approach

Requirement	Examples of initial user stories
Functional	
1. Spatial location of interventions	As a land manager, I want to see the spatial location of interventions so that I can decide where to implement 'water margin in arable field' SRDP AECS management option.
2. Information needs to be provided in a digestible format	As a land manager, I want the information to be provided in a digestible format so that I can decide where to implement SRDP AECS management options.
Non-functional /constraint	
3. It will be accessible for anyone to use.	As a land manager, I want to access the application from my tablet in a field.
4. It will be relevant and practical for land managers.	As a land manager, I want to explore relevant SRDP AECS management options for farm, so that I can understand the environmental benefits.
5. It will aim to be credible, with transparency in the information and methods used.	As a land manager, I want to be able to see the information and methods used in the web application.
6. It will be designed to be updateable with new information as it comes available.	As a land manager, I want the software to have the latest information on the SRDP AECS management options.

Web applications traditionally assume that the network is reachable. This assumption pervades the platform. This places web content at a disadvantage versus other technology stacks e.g. native mobile applications. Service worker are designed to redress this balance by providing a Web Worker[62] context, which can be started when prompted by user input. Service workers are JavaScript code that run separately from the main browser thread, intercepting network requests, caching or retrieving resources from the cache, and delivering push messages[63]. They are supported across a wide range of browsers[64]. Service workers depend on two main APIs to make an application work offline[65]: Fetch[66] (a standard way to retrieve content from the network) and Cache[67] (persistent content storage for application data). This cache is persistent and independent from the browser cache or network status.

4.1.1 Overview of server and client side rendering

The architecture most widely implemented, until recently, has been to use server-side rendering, which involves the browser fetching pages over HTTP[68]/HTTPS and it immediately returning a complete page with any dynamic data pre-rendered[69]. Advantages and disadvantages of server-side rendering include: it can provide a rapid first render; however, when reloading a page you throw away the entire DOM for each navigation and this is expensive when parsing, rendering, and laying

[62] https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API
[63] https://developers.google.com/web/ilt/pwa/introduction-to-service-worker
[64] https://jakearchibald.github.io/isserviceworkerready/
[65] https://developers.google.com/web/ilt/pwa/introduction-to-service-worker
[66] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
[67] https://developer.mozilla.org/en-US/docs/Web/API/Cache
[68] https://developer.mozilla.org/en-US/docs/Glossary/HTTP
[69] https://developers.google.com/web/ilt/pwa/introduction-to-progressive-web-app-architectures

out the resources on a web page each time; and it is a mature techniques with a wide range of tools to support it.

Client-side rendering is when JavaScript runs in the browser and manipulates the DOM. This provides advantages as page updates take place on the client, so that screen updates occur instantly when a user interacts with the page. Client-side rendering can selectively re-render portions of the page (or reload the entire page) when new data is received from the server or following user interaction.

It is common to render a page on the server and then update it dynamically on the client using JavaScript. The best practice is to combine server and client-side rendering, so that you first render the page on the server-side using data from the server directly; when the client gets the page, the service worker caches everything it needs for the shell[70] (interactive widgets and all). Once the shell is cached, it can query the server for data and re-render on the client (the rendering switches to dynamically getting data and displaying fresh updates).

4.2 Deciding how to implement requirements for map related functionality

There are a range of options when deciding to implement map related functionality in our application. Here we provide a summary of the most relevant options that have potential to meet the requirements (Table 3) e.g. free technologies that enable dynamic maps in web applications; these are OpenLayers[71], Leaflet[72] , and Mapbox GL[73] .

4.2.1 OpenLayers

Background and status

OpenLayers was originally developed by MetaCarta in 2005, it then became an Open Source Geospatial Foundation project in 2007[74]. In 2014 OpenLayers v3 was released to take advantage of new capabilities of modern browsers e.g. WebGL[75]. It is actively maintained with over 3000 stars and over 200 contributors on its Github repository[76].

Support

The main documentation is clear and well structured[77]; with links to a range of documentation[78] including a helpful 'quick start', tutorials (not as easy as Leaflet tutorials to use, due to large install requirements), workshop material, API docs and wide range of examples.

Functionality

OpenLayers is considered to have a wide range of functionality, often referred to as a full web GIS. It has 'draw' interaction, and the ability to use a range of vector and raster layers. One important

[70] https://developers.google.com/web/fundamentals/architecture/app-shell
[71] https://openlayers.org/
[72] http://leafletjs.com/
[73] https://www.mapbox.com/mapbox-gl-js/api/
[74] https://en.wikipedia.org/wiki/OpenLayers
[75] https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
[76] https://github.com/openlayers/openlayers
[77] https://openlayers.org/
[78] https://openlayers.org/en/latest/doc/

aspect of OpenLayers functionality is their use of the JavaScript Topology Suite (JSTS)[79], this library enables creation and manipulation of vector geometries and is a port of the widely used Java Topology Suite[80]. This is an advantage over other widely used JavaScript geospatial analysis libraries e.g. Turf[81].

Use in development and production

OpenLayers has moved to semantic versioning[82], so there is greater clarity when an incompatible/breaking API change takes place. OpenLayers is a large library, and to help developers only install the functionality they require they provide guidance on creating custom builds[83] (however this is not as a straightforward process). Advantages of OpenLayers include no need for an API key to use the standard functionality, and the tight integration with other open source geospatial projects e.g. OpenStreetMap[84].

4.2.2 Leaflet

Background and status

Leaflet was first released in 2011 by Vladimir Agafonkin, who then joined Mapbox in 2013 and he is still involved in its development[85]. It is actively maintained with over 21,000 stars and over 550 contributors on its Github repository[86].

Support

Like OpenLayers (and Mapbox GL) the main documentation is clear and well structured[87], with good tutorials and API reference.

Functionality

Leaflet provides basic map functionality. Additional functionality is provided through a rich ecosystem of plugins; however, you need to trawl through the long list of plugins to find those that are actively maintained[88].

Use in development and production

Leaflet is presented as "the leading open-source JavaScript library for mobile-friendly interactive maps. Weighing just about 38 KB of JS, it has all the mapping features most developers ever need"[89].

4.2.3 Mapbox GL

Background and status

Mapbox GL is based on Leaflet, and it uses WebGL to render interactive maps from vector tiles[90]. It is actively maintained with over 2700 stars and over 180 contributors on its Github repository[91]. It is

[79] https://github.com/bjornharrtell/jsts
[80] https://github.com/locationtech/jts
[81] http://turfjs.org/
[82] https://semver.org/
[83] https://openlayers.org/en/latest/doc/tutorials/custom-builds.html
[84] https://www.openstreetmap.org/#map=5/51.500/-0.100
[85] https://en.wikipedia.org/wiki/Leaflet_(software)
[86] https://github.com/Leaflet/Leaflet
[87] http://leafletjs.com/
[88] http://leafletjs.com/plugins.html
[89] http://leafletjs.com/

actively supported by Mapbox[92],a leading open source location data company that has grown rapidly since its origin in 2010- with a large amount of venture capital funding.

Support

The main documentation is extensive[93] (if a bit hidden amongst other products). It includes a useful range of examples and API reference. There are clear and structured lists of plugins, which are maintained by Mapbox[94]. In addition, there is a well written blog (covering a range of location products and services)[95].

Functionality

Mapbox GL was the first JavaScript web mapping library to make use of WebGL and vectors tiles[96]. Mapbox have driven the use of vector tiles, demonstrating the advantages (in most cases) over traditional raster tiles in terms of seamless navigation between zoom levels and rendering of labels. Mapbox has also created the leading map design tool called Mapbox Studio[97].

Use in development and production

Mapbox GL requires the use of an API key, especially for their vector tiles.

5. Current status and next steps

These three main mapping options are being implemented in a Node.js[98] and Express.js[99] based progressive web application. Express is a fast and lightweight web framework for Node.js. It is widely used in the development of progressive web applications. The next steps (Figure 1) are to continue working on challenges four (building a working web application), five (testing your software to ensure that it works as expected) and six (iterative and continuous development).

Acknowledgements

This report was funded by the Rural & Environment Science & Analytical Services Division of the Scottish Government. We would like to thank the interviewees and workshop participants who have supported this research.

[90] https://www.mapbox.com/mapbox-gl-js/api/
[91] https://github.com/mapbox/mapbox-gl-js
[92] https://www.mapbox.com/
[93] https://www.mapbox.com/mapbox-gl-js/api/
[94] https://www.mapbox.com/mapbox-gl-js/plugins/
[95] https://blog.mapbox.com/
[96] https://www.mapbox.com/help/define-vector-tiles/
[97] https://www.mapbox.com/mapbox-studio/
[98] https://nodejs.org/en/
[99] https://expressjs.com/

References

GULLIKSEN, J., GÖRANSSON, B., BOIVIE, I., BLOMKVIST, S., PERSSON, J. & CAJANDER, Å. 2003. Key principles for user-centred systems design. *Behaviour and Information Technology,* 22**,** 397-409.

HEWITT, R. J. & MACLEOD, C. J. 2017. What Do Users Really Need? Participatory Development of Decision Support Tools for Environmental Management Based on Outcomes. *Environments,* 4**,** 88.

MACLEOD, C. J. A. & HEWITT, R. 2017a. *Summary of research on developing a more integrated approach to land and water management using incentives and regulations for the delivery of multiple benefits: exploring national and regional level stakeholder views and needs.*: James Hutton Institute.

MACLEOD, C. J. A. & HEWITT, R. 2017b. *Workshop summary: developing an outcome-based approach for understanding the effectiveness of interventions in catchments for multiple benefits.*: James Hutton Institute.

MACLEOD, C. J. A. & HEWITT, R. 2018. *Developing an outcome-based web application: principles and requirements specification*. James Hutton Institute

MYERS, G. J., SANDLER, C. & BADGETT, T. 2011. *The art of software testing*, John Wiley & Sons.

VITOLO, C., ELKHATIB, Y., REUSSER, D., MACLEOD, C. J. & BUYTAERT, W. 2015. Web technologies for environmental Big Data. *Environmental Modelling & Software,* 63**,** 185-198.