Ecosystem infrastructure for smart and personalised inclusion
and PROSPERITY for ALL stakeholders

D203.3 Steppingstone applications for low cognitive and low ICT literacy

Project Acronym	**Prosperity4All**
Grant Agreement number	**FP7-610510**
Deliverable number	**D203-3**
Work package number	**WP203**
Work package title	**Collaborative development tools/ environments. T203.4**
Authors	**Steve Lee, OpenDirective**
Status	**Final**
Dissemination Level	**Public**
Delivery Date	**09/02/2017**
Number of Pages	**32**

Keyword List

Implementers, developers, end-users, user experience, low digital literacy, cognitive access, steppingstone, web, components, framework, guidelines

Version History

Revision	Date	Author	Organisation	Description
1	16/02/2015	Steve Lee	OPEND	ToC and Exec. Summary
2	17/03/15	Steve Lee	OPEND	1st version for review
3	16/12/16	Steve Lee	OPEND	2nd version ToC for review
4	24/01/17	Steve Lee	OPEND	For Peer review
5	25/01/17	Stefan Parker	KI-I	Peer Review
6	27/01/17	Daniel Ziegler	FhG-IAO	Peer Review
7	29/01/17	Steve Lee	OPEND	Final, ready for delivery

Table of Contents

List of Figures

Executive Summary

This deliverable provides development guidelines and the design of a starting framework of components for steppingstone applications designed for people with cognitive disabilities and low digital literacy. It documents the outcomes of background research and design activity carried out in Task T203.4.

The final aim of T203.4 is a set of open source components that integrate proven approaches and technologies for providing adaptive user interfaces suitable for those with low cognitive and digital literacy. These are being added to the Prosperity4All Developer Space to maximise availability and discovery.

In this context, **cognitive disabilities** include people with dementia and learning disabilities such as dyslexia. When we consider people with **low digital literacy** we largely refer to older people who have not grown up with modern digital technology and so are often extremely cautious and lack confidence. **Steppingstone** refers to the provision of configurable levels of complexity that may be configured to suite time variant user preferences. The features described here may be also suitable for other groups such as patients recovering from trauma like stroke but that is not discussed.

In order to get a sense of the state of the art, **Chapter 2** presents a review of some of the current work in research, standardisation and solutions. This is not intended to be an exhaustive literature review. Rather, it is included to inform the reader and provide background for the guidelines

The preferences of users targeted by this work are not yet well supported in main stream technology. Currently, accessibility standards and techniques have a bias towards sensory and motion disabilities. While there exist quite a few applications designed specifically for users with low cognitive and low digital literacy, there is an obvious need for more guidance and examples for developers. One specific need is that for users having a range of preferences around User Interface (UI) complexity, their abilities and so preferences may change overtime. For example, abilities slowly decline for those with dementia. Or elders become more confident with IT. **Chapter 3** in this deliverable explores these background issues.

Chapter 4 presents guidelines for developers when supporting these user preferences. We outline a number of effective principles that can be implemented in applications and reusable components.

Chapter 5 provides an outline of architecture, design and technology choices of the software components that are being developed to implement these guidelines.

The current status and future roadmap are outlined in **Chapter 6**.

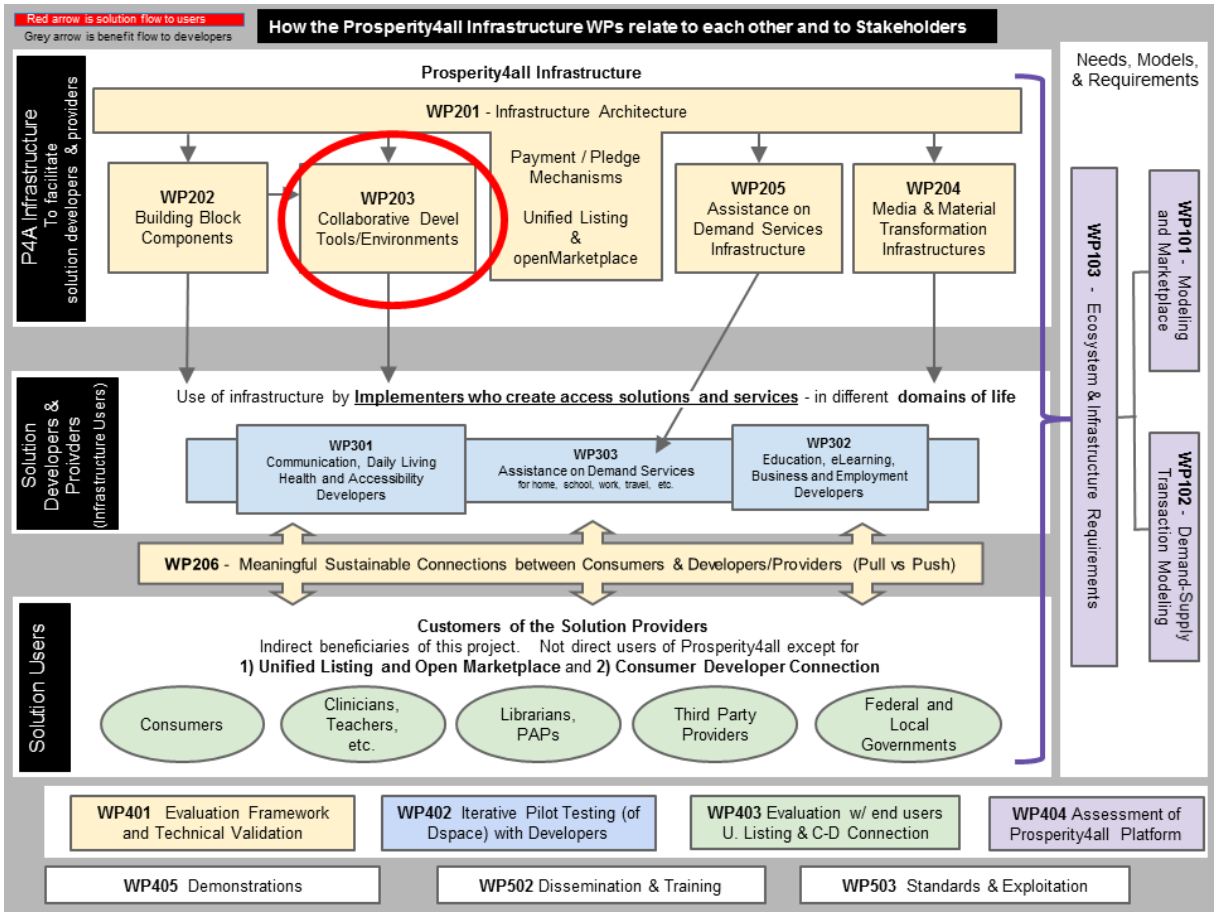
1 Contribution to the Global Architecture

The guidelines included here and the resulting component framework will be made available for reuse through the Prosperity4All Developer Space. The guidelines and framework are intended to make it easier for developers to both improve the low cognitive or digital literacy accessibility of their existing products and also create new technologies addressing these user preferences.

This work is in WP203 and is directly linked with the activities in WP201: infrastructure and Architecture and WP202: building block components. In addition, it is linked to WP401: Evaluation Framework and technical Validation.

In WP301 OpenDirective are enhancing Software as a Service (SaaS) product for people with low cognitive or digital literacy, their carer, and/or families. This work in Task T301.2 will use these guidelines and framework along with other outputs of Prosperity4All WP202 found in the Developer Space to enhance the user experience.

Figure 1 Overall Picture of Prosperity4All and WP203's position



2 State of the Art of Standards and Solutions

2.1 Research

There is a reasonable level of research[1] into the accessibility requirements of people with cognitive disabilities, especially age related. The latter is no doubt spurred on by the rapid rise in the number of people with Dementia. Some of this research has been used to develop prototype projects.

2.2 Standards that embrace Cognitive Accessibility

Accessibility standards, guidelines and support are strong in their support of sensory disabilities and perhaps to a lesser extent motor disabilities. Support for cognitive disabilities is currently low.

The key standards around which most activity exists on end user accessibility are:

- W3C WAI Web Content Accessibility Guidelines WCAG 2.0 [2]
- US Federal Government Section 508[3] (now based on WCAG)
- European EN 301 549[4] Accessibility requirements suitable for public procurement of ICT products and services in Europe

While each at least mention cognitive accessibility, the specific support is minimal. For example, in EN 301 549 “Some users will need the ICT to provide features that make it simple and easier to use” and in WCGA 2.0 “Text-to-speech software, which is used by some people with cognitive, language, and learning disabilities to convert text into synthetic speech”. Note that WCAG 2.1 will address this shortfall by including specific cognitive accessibility success criteria as explained below in section 2.1

Naturally, many of the recommendations in these standards are suitable for both sensory and cognitive accessibility but there is a lack of concerted focus on the preferences of these people.

Finally, there are other standards that are relevant, for example ISO/IEC 24751-2 Individualized adaptability and accessibility in e-learning, education and training, which defines a format for storing user preferences.

[1] http://www.clearhelper.org/resources/cwa/research/2014/
[2] http://www.w3.org/WAI/intro/wcag
[3] http://www.section508.gov/
[4] http://www.etsi.org/deliver/etsi_en/301500_301599/301549/01.01.01_60/en_301549v010101p.pdf

2.3 The Cognitive and Learning Disabilities Accessibility Task Force

This task force is part of W3C accessibility work carried out by the Web Accessibility Initiative and consists of members of WCAG and Protocols and Formats groups. Led by Lisa Seeman, the task force[5] has produced a number of documents and wiki in the last couple of years. It has progressed through a process of exploring web user requirements, analysing the gaps in current provision, identifying issues and developing techniques.

Initially the work was to provide advisory information for the WCAG group, but the task force is now working on providing a set of success criteria to be included in the 2.1 update to WCAG. This is much needed and will ensure WCAG addresses a more complete set of requirements.

The success criteria are being developed at the time of writing plus they are very detailed and quite prescriptive. So, while they will be extremely instructive and effective when WCAG 2.1 is released they do not meet the requirements for the guidelines outlined here. Thus, they have been used to inform the guidelines, along with other sources.

2.4 An Informal Review of Existing Solutions

When searching for existing ICT access solutions for people with cognitive or low digital literacy, it soon becomes apparent there is very limited choice. In contrast, various solutions accessibility tools are available for sensory and motion requirements. And while several previously specialist technologies like touch and speech interfaces have recently gone main stream, consumer user interfaces are still lacking in support for these groups.

Some support groups working with people with dementia have had success with selecting tablet apps for use in activities. It appears that the design and modality of touch interfaces on these devices works well in many cases. But almost none of the applications are the result of research into cognitive preferences or aimed at these users' specific preferences.

A few products exist to support elders with low digital literacy. These generally simplify the user interface for selecting tasks (eg smart phone launcher) and some also provide less complex versions of common application groups such as email. A good example is **Eldy**[6]. Others services such as **FinerDay**[7] provide a "walled garden" environment for this group of

[5] https://www.w3.org/WAI/PF/cognitive-a11y-tf/
[6] http://www.eldy.eu/
[7] https://finerday.com/

users, being separate from existing services such as Facebook. It's debatable whether this is a good thing compared to providing simplified access to such services.

Several academic projects also exist for people with dementia and other cognitive requirements. For example, **CIRCA**[8] presents a selection of images and videos designed to enable beneficial remanences. **Maavis**[9] is a framework application with a very simple and consistent user interface for people with dementia in supported environments providing access to media and video calls.

A recent European FP7 project working in the cognitive space is **IN LIFE**[10] which in part aims to find and improve projects and products for use by people with mild cognitive impairments or dementia and their carers.

Many of the those with low cognitive are supported by carers and in many cases, the carers also have low digital literacy themselves. However, no notable products exist that explicitly support this group's requirements.

[8] http://staff.computing.dundee.ac.uk/nalm/CC_website_at_Dec_2016/circa.html
[9] http://maavis.fullmeasure.co.uk/
[10] http://www.inlife-project.eu/index.php

3 Issues experienced by People with Low Cognitive Ability or Low Digital Literacy

3.1 Introduction to the User Preferences

As swedish accessibility service Funka Nu said in their March 2015 newsletter

> Cognitive accessibility – how come we know so little?

As explained in the preceding section, while there is good practical support for sensory accessibility via standards and APIs, there is much less for cognitive accessibility. Indeed, there is even a lack of basic research outputs, as Funka Nu discuss.

Part of the problem may be that the user base is so large and heterogeneous that it really can't be considered a group or category. Plus, there have not been the strong lobbying parties like those that say visually impaired users have had for years.

Having said that, there is a body of knowledge gathered by people working in the field and also new work in progress to analyse the gap between user preferences and provide techniques.

So, if this group is largely heterogeneous, what sort of common requirements are there? Well some of the areas people may have problems with include:

- **Memory** - Including: Working Memory, Short-Term Memory, Long-Term Memory, Visual Memory, Visuo-spatial Memory, Auditory Memory (memory for sound patterns and others).
- **Executive Functions** - Including: Emotional Control and Self-Monitoring; Planning/Organization and Execution; and Judgment.
- **Reasoning** - Including: Fluid Reasoning (logical reasoning), Mathematical Intelligence, Seriation, Crystallized Intelligence, and Abstraction.
- **Attention** - Including: Selective Attention, and Sustained Attention.
- **Language** - Including: Speech Perception, Auditory Discrimination, Naming Skills, and Morphosyntax.
- **Understanding Figurative Language** - Including: similes, personification, oxymorons, idioms, and puns.
- **Literacy** - Depends upon functions including: Speech Perception, Visual Perception, Phoneme Processing, and Cross-Modal Association (association of sign and concept).
- **Other Perception** - Including: Motor Perception, Psychomotor Perception.

- **Knowledge** - Including: Cultural Knowledge, Jargon (subject matter); Web Jargon and Technology; Metaphors and Idioms; Symbols Knowledge (such as icons); and Mathematical Knowledge.
- **Behavioural** - Including: Understanding Social Cues.

That's a very wide collection and some of these are also experienced by people with low digital literacy, especially those around knowledge. Due to the potential stress of using unfamiliar ICT these users may also be under cognitive stress or load, leading to other problems.

In order to more easily address common combinations of preferences, labels are often used for grouping users. Common categories include:

- Aging-Related Cognitive Decline
- Aphasia
- Attention Deficit Hyperactivity Disorder
- Autism
- Down Syndrome
- Dyscalculia
- Dyslexia
- Learning Difficulties
- Non-Verbal

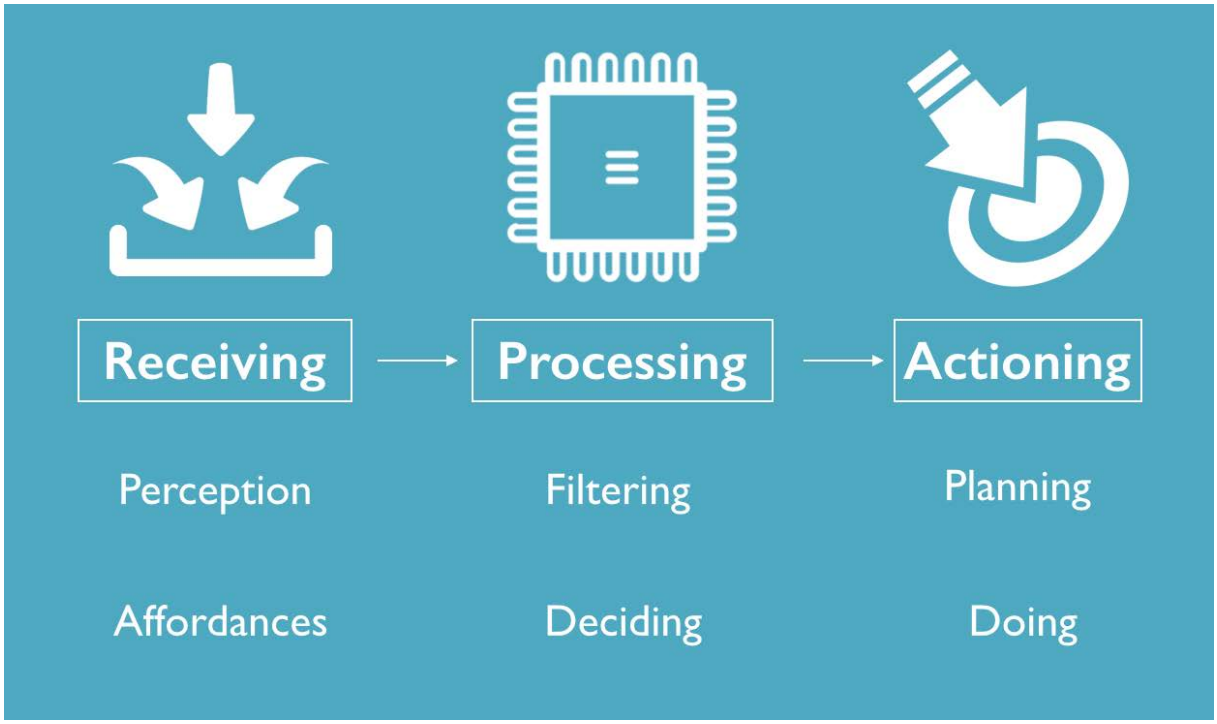
However, as we say in Properity4All: "one size fit's one". This is a way of recognising that each individual will have their own specific combination of preferences. Indeed, those we think of as having cognitive disabilities may also have sensory difficulties. Further, the exact depth or combination may vary over time. For example, as people get tired later in the day, when someone is recovering from a stroke or as dementia develops.

3.2 Example User Framework for Understanding Preferences

At CSUN 2015 Jamie Knight gave an informative talk on his personal experiences as a person with autism. In this he described a model he feels describes his particular approach to interacting with ICT [11]. This is very instructive. In essence:

[11] http://jkg3.com/Journal/cognitive-accessibility-101-part-1-what-is-cognitive-accessibility

Figure 2 A personal framework for approaching coga



1. Receiving
    - Perceptions
    - Affordances
    - Make as affordances (possible actions) as obvious as possible
    - Use language carefully, avoid idioms, literal terms and don't assume spatial knowledge
    - Make it easy to identify the most important thing.
2. Processing
    - Filtering
    - Deciding
    - Provide filtering tools eg via visual groups - uses find
    - Limit decisions - consequences - eg many ticket choices to choose from
    - Make it undo easy
3. Actioning
    - Planning
    - Doing
    - Avoid nesting if possible, make the plan easy
    - Provide feedback on processes, eg how far to go how much read
    - Inline help when whenever possible

4 Guidelines supporting Steppingstone Applications

4.1 Principles

In order to provide guidelines and a framework that developers can practically use in their own applications we need to select clearly obtainable principles. The following have been developed from various sources and chosen for wide impact but relatively easy implementation.

The principles have been chosen for impact and to be easily implementable by developers including being available in the framework components.

As mentioned above the W3C Coga task force are creating detailed success criteria (SC) to be included in WCAG 2.1. WCAG is required to be rigorous and developers will need guidance in how to implement the SCs and how to test they have been met. These principles are not as through as the SCs but are a step on the way.

- Non-intimidating
    - Welcoming
    - Clearly addressing user's reason for using
- Clarity
    - Affordances, language, focus on main thing
    - Low complexity with few distractions
- Clear Consequences
    - Easy navigation, clear and limited decisions required
    - Easy undo
- Orientation
    - Obvious location, feedback, contextual help
- User Preferences
    - Personal preferences including general accessibility

These principles address many of preferences of people with low cognitive and digital literacy. They should be kept in mind when designing website's and user interfaces.

4.1.1 Non-intimidating

This is obviously quite subjective and hard to test but any design should aim to welcome users in. Attempting to impress users with the owner's or developer's skills or portfolio is an antipattern.

It should be clear to the user that their needs are being addressed and they will be able to succeed in the task for which they have visited the site or are using the application.

4.1.2 Clarity

Users should be able to clearly see what they can do so as to reassure themselves. Language should be clear and direct as should controls that allow actions. Controls should be clearly actionable - that is they should have obvious affordances. Content should be clearly organised so that the main task a user can accomplish is clearly visible and preferably entirely on screen (unless it is multi step)

Complexity should be kept as low as possible as should distractions such as irrelevant content for the task in hand. For example, that means ads or moving content, unless it is clearly the main thing.

4.1.3 Clear Consequences

The consequences of the user actions should be crystal clear both before they are invoked, during system activity and afterwards. Navigation is included in this as well as actions such as ‘pay now’. Ideally at any point there will be only one or two easy decisions to be made.

Providing easy undo feature is an excellent way to reduce uncertainty and worry over unclear consequences.

4.1.4 Orientation

The user position in the user interface should be obvious at all times and clear routes to know locations should be available. Breadcrumbs are a useful way to achieve this. The same applies to any multi step process, clear feedback is required for what has already been done and how much is left.

Contextual help should be available to help orient the user when confusion occurs, don’t just provide a list of available actions.

4.1.5 User Preferences

Support of personal preferences is critical. In addition to allowing users to choose the accessibility options that suit them, cognitive options should be available. Fortunately, the Developer Space includes the Automatic Personalisation from Preferences (APFP) framework originally developed as part of the Cloud4All EU project.

The best way to determine which options are available is to ensure user involvement in all research, development and testing. For example, use the following questions addressed to actual users:

- What do they have trouble with?
- What tasks do they need help with?
- What tasks they avoid?
- What tasks often lead to mistakes?

4.2 Steppingstones

As mentioned before cognitive ability can vary over time. This may be cyclic or progressive, short term (e.g. during the day) or long term. For example

- Progressive Improvement
    - Elders who are new to ICT and developing new understanding and skills
    - Recovery from trauma such as stroke which causes temporary confusion
- Progressive Decline
    - Dementia exhibits gradual reduction in abilities, especially new memories
- Short term variation
    - Tiredness as day progresses can cause decline

In order to accommodate this variation a 'stepping stone' approach should be supported in user interface complexity. This provides several levels of cognitive ability and digital literacy with graduation between them.

For example, a photo viewer might have the following levels

- Simplest: view one photo at a time with a next button
- Medium: View several thumbnails at once with mechanism to view fullscreen
- Expert: Support albums and allow user to arrange photos amongst them plus add meta information

A mechanism should be provided in a user interface to adjust the level as part of a user's preferences, possibly managed by another user. This can be manual but support for some automation may be possible, especially as cognitive software services and AI gain popularity.

Figure 3 Mockup of low complexity photo viewer

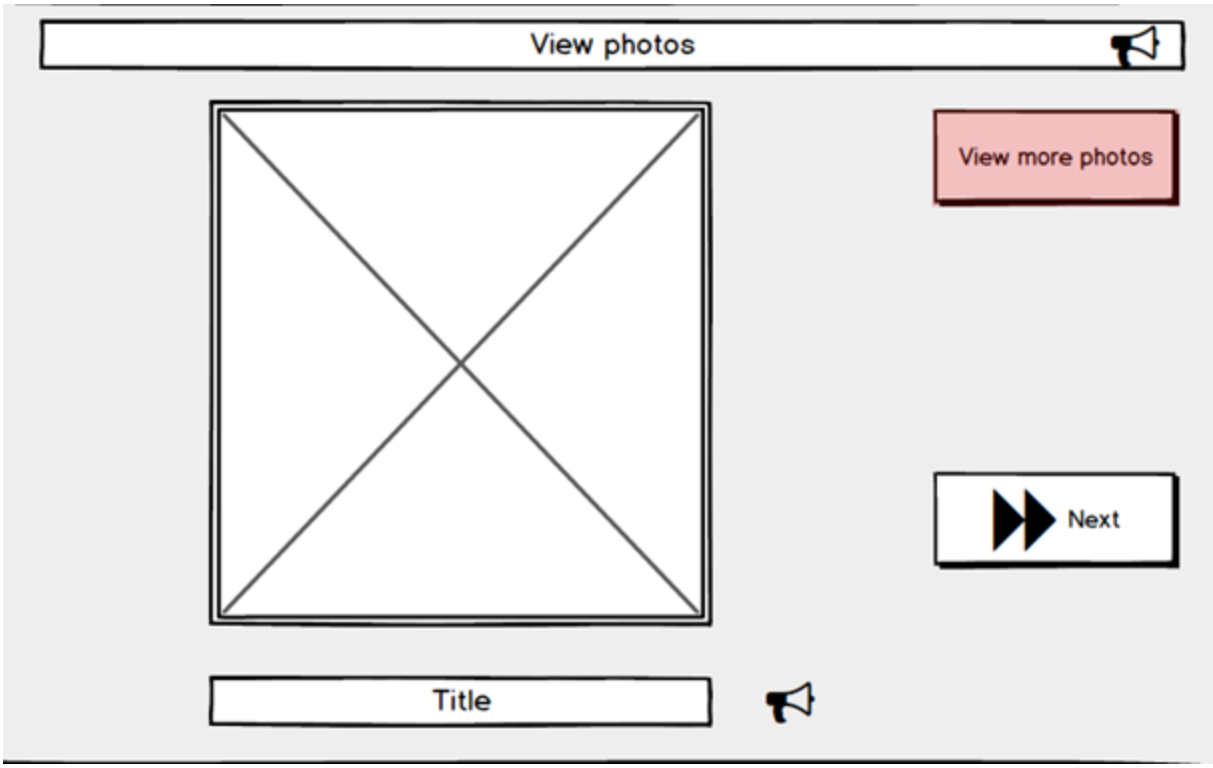


Figure 4 Mockup of medium complexity photo viewer

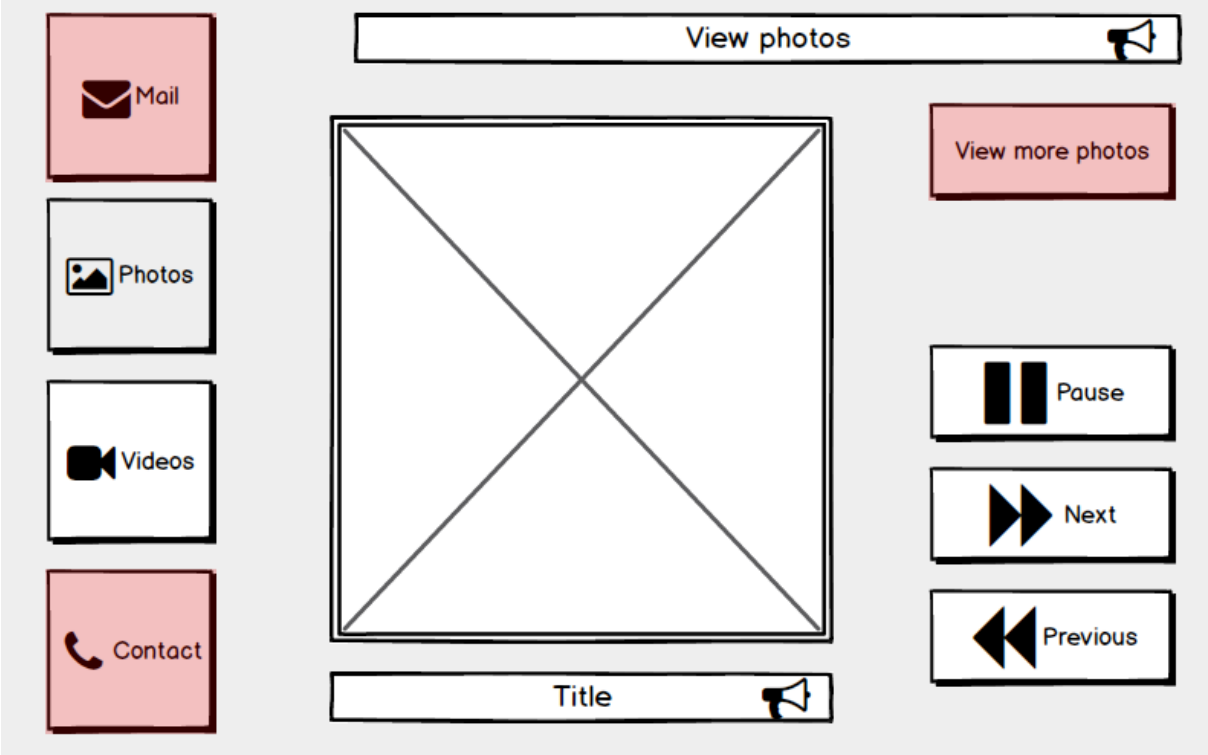
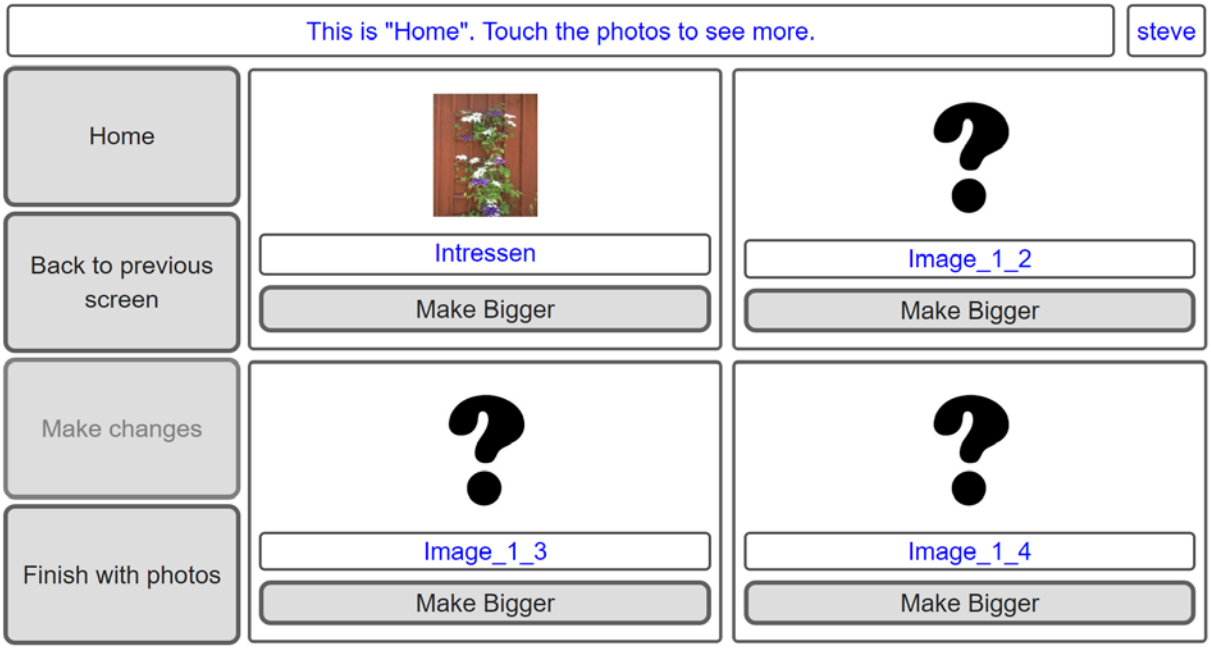


Figure 5 Prototype of high complexity photo viewer with edit facilities



5 A Framework of Software Components

5.1 Concepts

OpenDirective are working to present a set of reusable components in the Prosperity4All Developer Space. These will embed the principles outlined in the previous chapter. Developers can then use them in their own applications or to explore the principles through user engagement and research.

It is also hoped these components will act as hub for exploring new principles for best supporting people with low cognitive and low digital literacy. In other words, we hope to encourage open collaboration to increase knowledge of user's preferences and how to best support them. For example, the growing availability of Artificial Intelligence for support should be a promising area to explore.

The components need to be usable by a wide range of developers and researchers. They also need to encourage open innovation with developers, researchers, users and their representatives. Finally, the components should be readily usable in commercial products, hopefully with open collaboration to improve the components.

The collaboration goal is best realised through open source collaboration. Discovery and engagement are at the heart of the Developer Space goals so it is a natural fit for the components to be held there. In reality this means the components will have their own external repositories and community that are linked to from the Developer Space using its tools designed for this purpose. Thus, developers with an interest will be signposted to the components.

5.2 Goals for a Framework of Components

We need to define what we mean by 'components'. The basic idea is an encapsulated chunk of user functionality that can be readily dropped into any application. In reality, the idea of a component can apply at several scales. For example, we could consider simple controls such as a colour picker. Or at the other extreme we might be thinking of a complete applet used in a dashboard or full screen.

For the steppingstone components, our aim is somewhere in the middle of this range. Our components are distinct activities that can be encapsulated in the UI, the code and related backend logic. For example, a photo viewer displays photos in various formats and with several levels of UI complexity. It uses clearly defined but abstracted APIs to get the images to display and example backend code to implement them is provided.

Interfaces are provided to let the components talk to each other, but in a loosely coupled way, through messages.

The specific details will become clear as the implementation continues.

5.3 Broad Principles

5.3.1 Web Technology

Web technologies are having the widest reach across platforms and devices which is why they were explicitly mentioned in the DoW. Properly designed web components have the potential to reach users across the world, from those using small cheap under powered phones with hopeless mobile connectivity to users on large powerful desktops computers with always on fast fibre internet connections.

In reality it is hoped the components will be used in real solutions for users who might be using a wide range of technology and in varied contexts. A typical use in the western world could be in residential situations with mostly working WiFi. Devices could be tablets or smart TVs.

Developers can use web technologies to create apps that users access directly in web browsers. Alternatively, so called hybrid apps wrap a browser in a native app in order to access platform features. It should be noted that the pace of browser support for new web standards is rapid. As a result, the gap between native and web apps narrows as the latest browsers can access many platform features such as contacts or audio. In addition, browsers themselves now provide advanced, computation intensive APIs such as Text to Speech and audio.

5.3.2 Progressive Web Apps

An advantage often cited for native apps is that users prefer being able to get them from app stores rather than enter a URL into a browser. The reality is less straightforward and web apps more are popular in some cases. Google's Progressive Web App (PWA)[12] concept supports a mechanism where users can pin websites that are used repeatedly as an icon on the desktop / launcher. They will then be presented full screen without the browser UI (so called Chrome). PWAs also support offline work and caching of resources.

Our components will support being run in PWAs.

[12] https://developers.google.com/web/progressive-web-apps/

5.3.3 Progressive Enhancement

The wide range of user devices are addressed by two key techniques. Progressive Enhancement (PE) aims to provide a user the best possible experience given the technical capabilities of their device. The alternative is to design for a lowest common denominator, or worse, only work on the most capable equipment.

A number of techniques can be used to ensure users on low powered devices or with poor connectivity still get a usable experience. For example, it's common to use so called serverside rendering to HTML to create a basic static HTML web page and then add dynamic features through JavaScript on the device as its capabilities enable. PE can involve more work than solutions that assume full JavaScript capabilities, as many current frameworks do but it's a question of user convenience verses developer convenience.

Interestingly, the steppingstones concept outlined above could be viewed as being a user version of PE.

5.3.4 Responsive Design

The other technique to support widely varying devices is Responsive Design[13] which aims to provide a good experience on a wide range of screen sizes. The idea is to alter the content and layout of the UI according to screen size so as to make the most of available space.

There is also often an assumption that small screens are general smart phones so are touch enabled and not accessed with mouse and keyboard. Smart developers minimise such assumptions, especially as touch devices can have Bluetooth keyboard added and many two-in-one "laptops" have touch screens.

5.3.5 Ethical Web Development

Both of these techniques boil down to 'don't make assumptions', a principle at the very heart of creating accessible interfaces too. These ideas and more are neatly encapsulated in the Principles of Ethical Web Development by Adam Scott[14]

The components will follow this excellent ideal.

[13] http://alistapart.com/article/responsive-web-design
[14] https://ethicalweb.org/

5.4 Programming Models and Dependencies

Next comes the tricky question of which programming models and frameworks or libraries will be used to implement the components. The 2 are treated together here as they are often closely related with toolkits implementing a particular model. We'll look at tooling used to build them in the next section

We have several goals here

- Only use models and dependencies that add significant advantage over 'raw' JavaScript manipulating the Document and Browser object models (DOM and BOM).
- Choose techniques and technologies that are well supported, reasonably popular and easy to learn.
- Minimise dependencies so the components can be used in a wide range of projects without clashes.
- Keep the resulting code size small so as not to be an issue in mobile situations.
- It hardly needs to be said but – 1st class accessibility must be supported
- Open source is a must, and a preference for code that will be in the Prosperity4All Developer Space

The web development framework and tooling space is undergoing a breakneck pace of development recently. New ideas and tools come out all the time and 'the best' techniques change rapidly. For example, at the time of writing React and Redux are hugely popular, as is Angular 2 and Vue is a rising star. But they were only just appearing when we started selected technologies for the steppingstone components. As well as these frontend, focussed tools, backend architectures are evolving with cloud and microservices becoming mainstream.

We obviously shouldn't chase the latest "shiny" but also do not want to exclude effective tools or use any reaching the end of lifetime. Researching tools to meet the requirements was a substantial activity but we had to "draw a line" at some point. We made a few choices that at the time seemed a little risky but they have proved good as they steadily have gained in popularity. For example, a recent O'Reilly article calls out some of our choices as trends for 2017[15]

[15] http://www.dataarchitect.cloud/5-web-trends-for-2017/

5.4.1 Components

The complexity with components is in the exact form they take, dependencies and minimising clashes with the using app's HTML, CSS and JavaScript.

W3C Web Components[16] [17]would be ideal as they will be directly supported by browsers. The problem is after many years they still have not landed, though some parts of the specification are now supported fairly widely (but obviously not in old browsers)[18].

Thus, we need to use another specification. React or Angular are currently the most popular front end tools to support components, with Vue catching up fast. However, React and Vue do not do that much as they are small libraries which really only offer the view part of a typical Model View Controller (MVC) architecture. At the other extreme, Angular is a very large and highly opinionated framework. React addresses the view part of a MVC design, which is why other techniques like Redux are popular for managing state in large apps.

It's worth keeping in mind that these tools from the likes of Facebook are addressing the needs of large complex front end application teams which are unlikely to be the same in smaller apps using the steppingstone components. They are certainly largely overkill for the components themselves.

A final piece of the puzzle is that modern applications, and so the components orchestrate a lot of asynchronous activities. These use events, requests to servers and APIs. We wish to use messages between the components to give loose coupling and even local file access will be async. While JavaScript is essentially asynchronous with a single message thread, JavaScript and imperative coding techniques have not made handling easy for developers.

Traditionally, callbacks are used to handle asynchronous behaviour but these soon become hard to manage as complexity grows. This leads to the well-known 'callback hell' of deeply nested hard to read code. Newer incarnations of JavaScript support Promises and Async/Await which help dramatically, though browsers do not yet support them (but seen tooling below).

5.4.2 Reactive Programming with ReactiveX and Functional programming

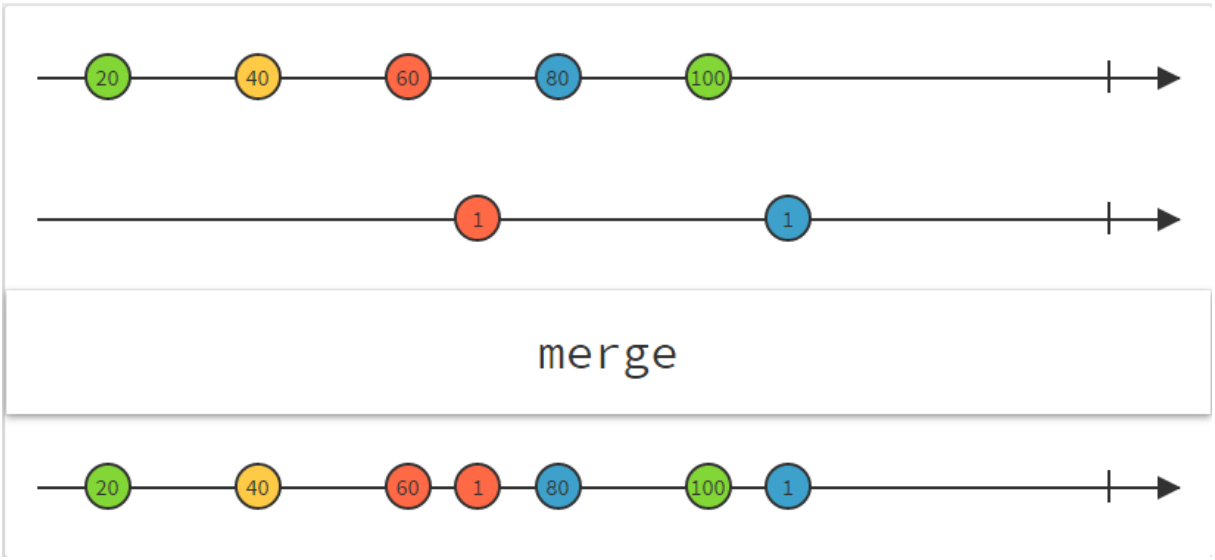
A workable solution is Reactive Programming[19], sometimes called stream based programming. This is a programming model oriented around dataflows and the propagation

[16] https://www.w3.org/TR/custom-elements
[17] https://www.w3.org/TR/shadow-dom
[18] http://caniuse.com/#search=web%20components

of change so fits perfectly with asynchronous development. ReactiveX[20] (Rx) is an implementation from Microsoft that is around 6 years old. The JavaScript version, RxJS, is also mature, version 5.0 having just been released.

Rx is a combination of the Iterator and Observer design patterns and as a result code becomes declarative operations on streams, rather than imperative logic. Small functions are used to declare ways changes are made to the data flowing through the code (eg map or reduce). This has significant advantages but does impose a learning curve for developers. That said there's an excellent introduction here[21].

Figure 6 An example Reactive data stream operator shown as a marble diagram



ReactiveX code also uses Functional Programming (FP)[22] techniques, but without all the highly theoretical and more complex aspects that often come with FP. Thus, developers who are familiar with class based OOP and imperative code may feel alienated for a while. However, the benefits for code clarity and quality are substantial so we use it in the steppingstone components. Interestingly Angular 2 has adopted RxJS so it is becoming better known amongst developers.

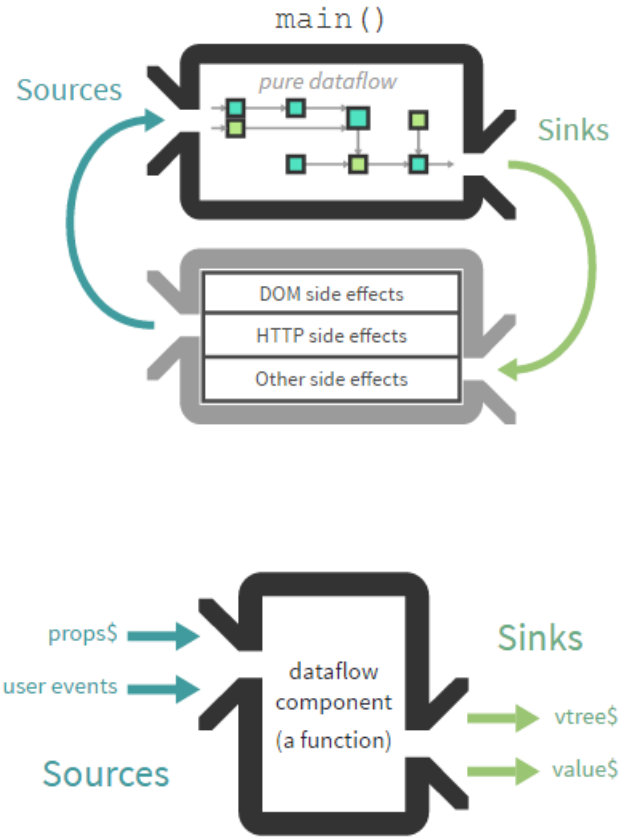
[19] https://en.wikipedia.org/wiki/Reactive_programming
[20] http://reactivex.io
[21] https://gist.github.com/staltz/868e7e9bc2a7b8c1f754
[22] https://en.wikipedia.org/wiki/Functional_programming

5.4.3 CycleJS

CycleJS[23] is a tiny library that offers an opinionated way to use RxJS. Its strength is clean separation of side effects in 'drivers' from the pure functional 'business' logic. This is a powerful and clean way to organise code and it directly supports a simple component model[24]. Having the logic implemented as pure functions with classes holding state makes testing much easier: you just provide inputs and test for expected outputs. Not complex mocking of objects.

While CycleJS started out using RxJS it now has its own RP library called xstream[25]. This is designed specifically for use in CycleJS applications and removes some of the complexity of Rx. Incidentally, the author of CycleJS and xstream is Andres Stalz who is a major contributor to the RxJS project.

Figure 7 A cycle application and component



[23] https://cycle.js.org/
[24] https://cycle.js.org/components.html
[25] http://staltz.com/xstream/

5.5 RESTfull APIs and GraphQL

The components will need to talk to backend logic and RESTful[26] APIs are the natural way to do this. This has been the preferred approach used in web services and SOA. Swagger[27] is used to define and document the APIs, plus it allows tooling to provide intelligence when creating and using RESTful APIs.

One place where REST APIs get complicated and slow is when the APIs returned data comes from a number of sources. This can get expensive to manage and so Facebook have developed GraphQL[28] as a query language for APIs. It is becoming popular and has the advantage of putting the client in charge of selecting what data it needs and so reducing bandwidth compared to REST where all data is returned as decided by the server. This will be used where appropriate.

5.6 Serverless Architecture for the Backend

While the components are predominantly frontend running in browser, there will always be some backend requirements. In addition to server side rendering required for Progressive Enhancement, various services for the frontend will be required. Some will be part of the Framework, others will be external 3rd party services. The backend itself will call various services.

Nodejs[29] is a mature and popular open source server technology with a large open source ecosystem. The main benefit is it is also JavaScript so somewhat reduce developer entry requirements. It is proven in web servers, API servers, Microservices and many more .In addition, many open source projects now use it for developer tooling as well as backend.

Ideally, rather than a monolithic backend we want to break it up so each component comes with its own backend logic. These can then be combined with others for build out an app. This leads to a microservice architecture. Though, as the code is open source developers will be able to use it a monolith if they desire.

[26] https://en.wikipedia.org/wiki/Representational_state_transfer
[27] http://swagger.io/
[28] http://graphql.org/
[29] https://nodejs.org/en/

We really don't want to conflate the backend logic with complex deployment and DevOps requirements. One approach considered was to be to provide code fragments that can be plugged into a node server such as Express (the 'standard') or otherwise used by developers in their own backend deployments.

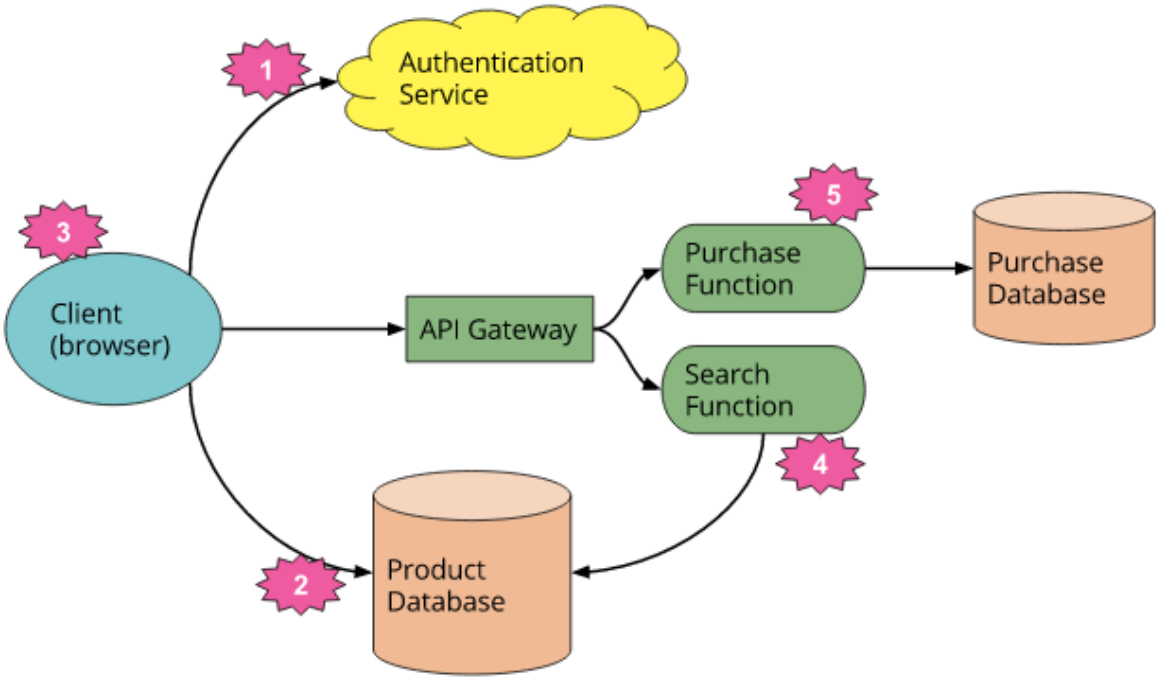
Fortunately, however, a fairly new backend architecture has arrived on the scene that fits our needs perfectly.

Serverless architecture[30] is an evolution of Platform as a Service (PaaS) cloud provision that allows developers to concentrate on business logic. The cloud provider manages all the lower levels of the stake; hardware, VMs, containers, operating system, web servers etc. A Serverless developer provides a fragment of code that is executed in an event driven manner. They then configure how it will be invoked and connections to external services. HTTP triggers are available as are message queues and data storage updates. Bindings are provided to cloud services such as data and storage.

Amazon introduced this concept to developers a couple of years ago, by launching their Function as a Service (FaaS) Lambda feature of AWS. Now, other cloud vendors are falling over themselves to provide their own Serverless solutions. It's still a young architecture with plenty of wrinkles but OpenDirective visited the recent London Serverless Conference[31] and the presentations[32] convinced us that it is the right approach for the Steppingstone components.

[30] https://www.martinfowler.com/articles/serverless.html
[31] http://london.serverlessconf.io/
[32] https://www.youtube.com/channel/UCqlcVgk8SkUmve4Kw4xSlgw

Figure 8 Example of a serverless architecture (Martin Fowler)



We have selected Microsoft Azure Functions[33] as the most appropriate solution for the Prosperity4All Steppingstone components, not least as vendor lock-in is reduced by much of the code being open source[34].

5.7 Tooling

As with the code, for developer tooling we want to keep to low dependencies and simplicity. We also require good development techniques including Unit tests.

Nodejs again delivers an excellent open source solution that meets our requirements. In order to keep build tooling simple, we use npm build scripts rather than the more complex Grunt or Gulp.

For unit testing we use Tape[35] which is lightweight and compatible with many tools.

Although any editor can be used for the code we take advantage of Visual Studio Code[36] as a cross platform lightweight editor with great features for web development with node.

[33] https://azure.microsoft.com/en-us/services/functions/
[34] https://github.com/Azure/Azure-Functions
[35] https://medium.com/javascript-scene/why-i-use-tape-instead-of-mocha-so-should-you-6aa105d8eaf4#.6i43eecmy

Modern front end development uses modules and requires bundling tools to manage static assets including a single JavaScript file to be included by the browser. We selected WebPack[37] for being highly flexible and having declarative configuration. It is very popular with the large React community so is well supported.

The latest version of JavaScript, ES6, includes several features that significantly make development easier[38], especially with Reactive Programming (eg arrow functions, destructuring and rest/spread operators). However, while Nodejs supports much of ES6, modern browsers do not. Plus, we need to target older browsers. The solution is to use Transpilation down to ES5 as part of a build process. ES5 is very well supported by browsers. While Babel is very popular for transpilation, the TypeScript[39] language also works well and has another advantage for components and libraries.

Typescript is a superset of JavaScript that adds typing to JavaScript. It is a compiler that outputs ES5 JavaScript. While typing may not be as useful for improving code quality in smaller projects it does allow interfaces to be defined which improves documentation. But the real benefit for developers using libraries as components is that the compiler adds both explicit and inferred type information that editors can use to greatly enhance the developer experience. Visual Studio code calls this intellisense[40] and it means the editor will often provide choices and highlight errors.

Finally, running the code through a “lint” checking tool is an important step for ensuring code quality. TSLint[41] for TypeScript is not as advanced as ESLint for JavaScript but is adequate. If any ESLint features are required, it is possible to run them on the JavaScript generated by the TypeScript compiler

[36] https://code.visualstudio.com/
[37] https://webpack.github.io/
[38] http://es6-features.org/#DefaultParameterValues
[39] http://www.typescriptlang.org/
[40] https://code.visualstudio.com/docs/editor/intellisense
[41] https://palantir.github.io/tslint/

5.8 Summary

As this list highlights, web development is no longer a simple matter of edit some code and refresh the browser. Quite a few techniques, technologies and tools are required to build modern web apps.

5.8.1 Principles

These are largely encapsulated in Ethical Web Development and Reactive programming

- Accessibility
- Progressive Enhancement
- Progressive Web App
- Reactive Programming
- Functional Programming
- Loosely couple components
- Serverless backend logic
- RESTFull APIs, GraphQL if it makes sense.

5.8.2 Technologies.

Remember we want minimal dependencies and simplicity to minimize the barrier to use.

- Front End
    - TypeScript - compile to Javascript to use ES6 and provide intellisense
    - CycleJS
    - Xstream – stream library for CycleJS
    - Lodash for functional utilities
- Backend
    - Microsoft Azure Web Apps and Functions
    - Nodejs - javascript
    - Static web content
- Tooling
    - Visual Studio Code and Command line
    - Nodejs
    - Npm scripts for build
    - TSLint for quality
    - Tape for tests
    - Webpack for bundling
    - Chrome Developer tools

6 Current Status and Roadmap

The selection of the previously listed technology stack has taken a considerable time with many examples having been explored and software “spikes” made to explore feasibility. We are now developing the components and refining the component model and how they interact in both the UI and backend.

An early mockup using some of the technologies is available in GitHub in the OpenDirective/steppingStones repository[42]. This does not have any backend or distinct components yet.

Looking ahead, we will be providing a number of components in the above repository and also in the Prosperity4All Developer Space by the end of the Prosperity4All project. Some of these will be included in the enhanced Brian product being worked on in T301.2, along with other SP2 elements.

The components are available under the permissive MIT open source licence

All steppingstone project tracking is on GitHub but here is a list of planned components.

- Photo viewer
- Email viewer
- Video viewer
- Video calls

[42] https://github.com/OpenDirective/steppingStones

Annex I: Glossary

Abbreviation	Full form
AoD	Assistance on Demand
API	Application Program Interface
AT	Assistive Technology
C4A	Cloud4All
D	Deliverable
DoW	Description of Work
DSpace	DeveloperSpace
GUI	Graphical User Interface
GPII	Global Public Inclusive Infrastructure
ICT	Information and Communications Technology
ISO	International Organization for Standardization
IT	Information Technology
KPI	Key Performance Indicator
P4A	Prosperity4all
R&D	Research and Development
REST	Representational state transfer – a web-native protocol
SP	Sub-Project
Transpilation	Converting one version of JavaScript to another (older) version
UI	User Interface
UX	User Experience
WP	Work Package
W3C	World Wide Web Consortium
WAI	Web Accessibility Initiative
WCAG	Web Content Accessibility Guidelines