Curtin HIVE (Hub for Immersive Visualisation and eResearch)
and State Library of Western Australia

Developing an Interactive Kiosk Version of OldPerth

Nidup Dorji

Supervisors: A/Prof Andrew Woods (Curtin HIVE), Barbara Parnaby (State Library of Western Australia), Debra Jones (State Library of Western Australia), Michael Ovens (Curtin HIVE)

Curtin University, GPO Box U1987, Perth WA 6845

February 2023

A research project supported by the Curtin HIVE Summer Internship Program (2022-23)

Nidup Dorji, A/Prof Andrew Woods, Barbara Parnaby, Debra Jones, Michael Ovens
"Developing an Interactive Kiosk Version of OldPerth"
Curtin HIVE internship report (2023).

Abstract

The State Library of Western Australia (SLWA) has an extensive collection of photographs of Perth and its surrounding areas within the state. Previously, projects have developed a website called OldPerth which hosts over 12,000 images from SLWA's collection to be browsed on an interactive map for user exploration.

While the OldPerth website offers an engaging user experience, its suitability for use on public consoles in the library is restricted due to an abundance of external links, limited interactive features, and the potential for users to leave the map in a vacant area of the map. The primary objective of this project was to develop a self-contained and interactive kiosk-based version of OldPerth which provides an enhanced user experience of the State Library of Western Australia's extensive historical photograph collection.

The kiosk version of OldPerth addresses the limitations of the current OldPerth platform, rendering it suitable for deployment on public consoles. This is achieved by replacing external links with QR codes, introducing interactive features that randomly select locations and scroll through photographs to engage users, and implementing an auto-reset mechanism to reset the site after prolonged periods of inactivity. This will allow the State Library to make more use of OldPerth at their city location, and hence provide more visibility for their collection.

Table of Contents

1. Introduction

The State Library of Western Australia (SLWA) has an extensive archive comprising over 700,000 media contents of Perth and various regions within Western Australia. These are systematically catalogued within the library using the Machine-Readable Cataloguing (MARC) file format [1].The MARC file contains metadata relevant to the media, including details such as photographer, subject, location, and description. However, the MARC files have become outdated and pose considerable usability challenges. Notably, end-users face difficulties in accessing the large media repository of the State Library through text searches due to the intricate interconnection of MARC files or potential omissions within them [2].

Due to challenges associated with accessing the MARC files, Max Collins, a former intern, proposed a solution. Collins developed the LibClean, a library that enables the extraction of metadata from MARC files and use the metadata to geolocate each image [2]. The resulting images are made publicly accessible through an interactive map. However, the current user experience is confined to the website and is not deployable on public consoles due to the presence of numerous external links and lack of interactive feature that would potentially increase user interactions.

The main aim of the project was to build a self-contained, user-centric and interactive experience on the OldPerth website for individuals physically visiting the State Library in the Perth Cultural Centre. The kiosk version of OldPerth provides an alternative means of exploring the State Library's historic photograph collection. To achieve this overarching goal, the project comprised of four sub-goals:

1. Replace embedded links on OldPerth website with QR codes to enhance security and privacy.
2. Develop and integrate an interactive feature on the website to autonomously select a location, browse images, and reset website after a set period, enhancing visitor engagement and interaction.
3. Investigate methods for incorporating other formats of heritage collections into the website, with a specific focus on audio-visual content such as films or oral histories.
4. Review codebase of the site and update the site with the latest branding and relevant information.

Over the course of the project, several new features were developed: a QR code to replace hyperlinks, an auto-passive interactive feature, and auto-reset mechanism that resets the website. These new features culminated led to the creation of OldPerth kiosk version, which inherits the similar functionalities from OldPerth website but optimised for consoles.

This project was made possible by technical support from the Curtin HIVE Summer Internship Program (2022-23) and financial support from the State Library of Western Australia. The Curtin HIVE Summer Internship Program allows a Curtin student to undertake a 10-week full-time internship to undertake a research project investigating the application of visualisation technologies in a particular discipline area. Interns were supervised by an academic discipline leader over the course of the 10 weeks. The students had regular access to the Curtin HIVE facility and were supported by HIVE staff. The results of the HIVE Summer Internship projects were presented and demonstrated at the HIVE Summer Intern Showcase held at the Curtin HIVE on Friday, 2nd of February 2024, and recorded in a written report.

2. Background

Mark Shelton's Thickshake

Mark Shelton, a former intern, worked on enhancing the SLWA's library cataloguing through an application of modern computing power [3]. Over the course of the project, he developed a Python package, Thickshake, that reads the MARC codes and converts them into a relational database [3]. Shelton believed that using a relational database for a library catalogue enables faster readability, editing, and reporting [3]. Thickshake is not limited to a relational database but can also produce data in other formats (CSV, JSON, HDF5). This versatility enabled running the OldPerth website using JSON data format.

Thickshake has a parsing framework that contains multiple parsers. One of these parsers is the meta parser, which reads the image comment field (856z). It processes the relevant information from comment field ("856z) and saves it to the database. Additionally, the parsing framework includes a geolocation parser, responsible for parsing metadata in the field (856z) and converting it into GPS coordinates. Shelton employed a loose regex pattern and dictionary matching to extract an address in the form of "street number street_name street_type, suburb_name" from the comment field. Once extracted, the Mappify.io API (Application Programming Interface) was utilised to determine latitude and longitude coordinates based on the structured addresses [4]. Two other parsers, namely the date parser and image parser, serve distinct purposes. The date parser extracts dates from the note field (856z) and date ranges from the date range field (100d). Simultaneously, the image parser efficiently extracts image dimensions from image URL fields (856u) without the need to download.

Thickshake also featured image processing functionality, categorized into face detection and text recognition. Face detection involved identifying and tracking people of interest, employing Python libraries such as Dlib-ml [5] face detector and OpenCV [6]. The text recognition aspect focused on extracting information from embedded text or content in the image, utilising Google's Tesseract library [7] and OpenCV's character recognition tools through multiple processing steps. Shelton advocated for a machine learning framework over per-record basis parsing. Consequently, a machine learning function for subject recognition was developed by combining face detection results with metadata from the catalogue.

Max Collins LibClean

Another former intern, Max Collins, completely redevelop the framework for cataloguing the library by developing a python package known as LibClean [2]. The main goal of the LibClean was to simplify the geolocation of images based on their metadata. LibClean takes the MARC 21 formatted XML file and parse the metadata using the python package Pymarc [8] previously utilised in Shelton's Thickshake. It extracts relevant information from comment field (856z), the URL field (856u) and the date field (260). Once the metadata has been extracted, a photo data object (pho_data) is constructed with relevant fields.

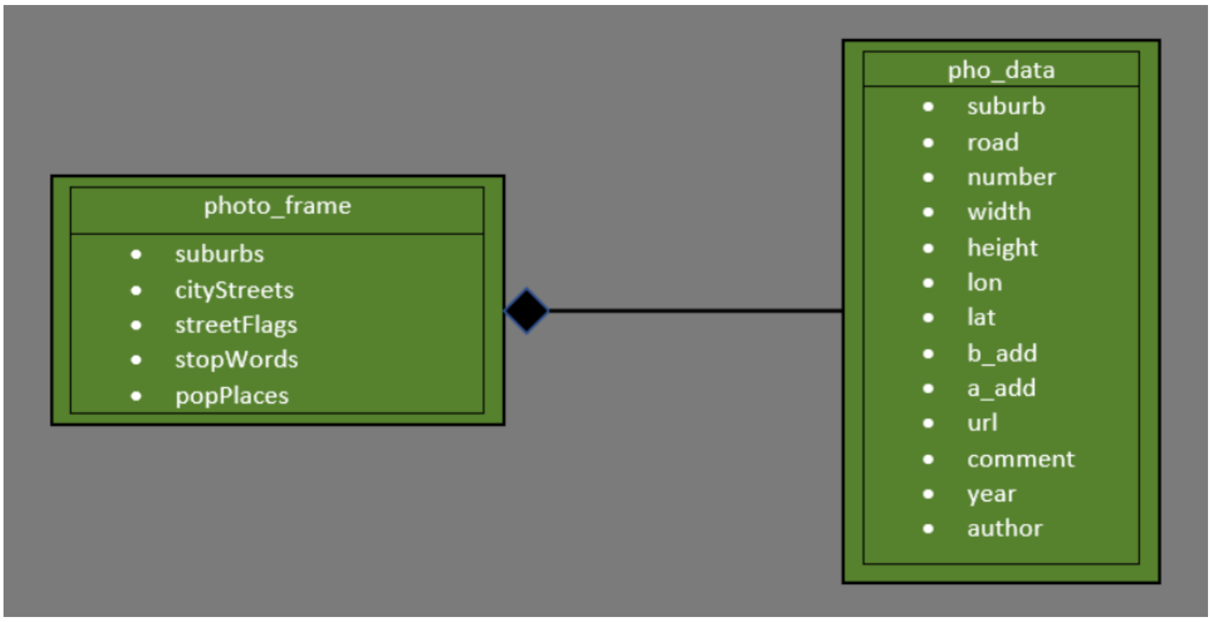


Figure 1. *LibClean UML Diagram [6]*

The photo data object undergoes continuous updates through the output generated by a series of parsers. These parsers employ a combination of regular expressions and dictionary matching. Executed in the order of perceived accuracy, the parsers aim to identify addresses associated with popular locations using the "__pop_sweep" parser, streets within Perth city with the "city_street_sweep" parser if the "__pop_sweep" parser fails to find a match, and more lenient street names with the "suburb parser," ultimately determining the suburb for each address.

The current OldPerth website operates without structured addresses, relying on GPS coordinates to locate images. LibClean, like Shelton's Thickshake, continues to use the Mappify API and it converts structured addresses into latitude and longitude coordinates. Unlike Thickshake, LibClean can perform checks on the returned addresses from Mappify to validate the accuracy of geolocation results, ensuring they match the initially sent addresses. Invalid addresses are labelled to prevent redundant queries.

LibClean introduces an additional functionality to correct the geolocation of images through user inputs. A Google form is integrated into the website, allowing users to submit the correct address for an image. The data is stored in Google Drive in CSV format and can be downloaded for parsing by LibClean, which generates new latitude and longitude coordinates for the images. To prevent data loss during the time-consuming geolocation process, LibClean incorporates the ability to save and load image catalogues from Google Drive using the Python JSON package [9].

LibClean also feeds the geolocated data back SLWA catalogue, and this ensures that both OldPerth and SLWA have validated addresses. Additionally, to integrate to the OldPerth site with new coordinates returned, Collins developed the OldPerth-client, enabling to update the OldPerth site without requiring programming skills.

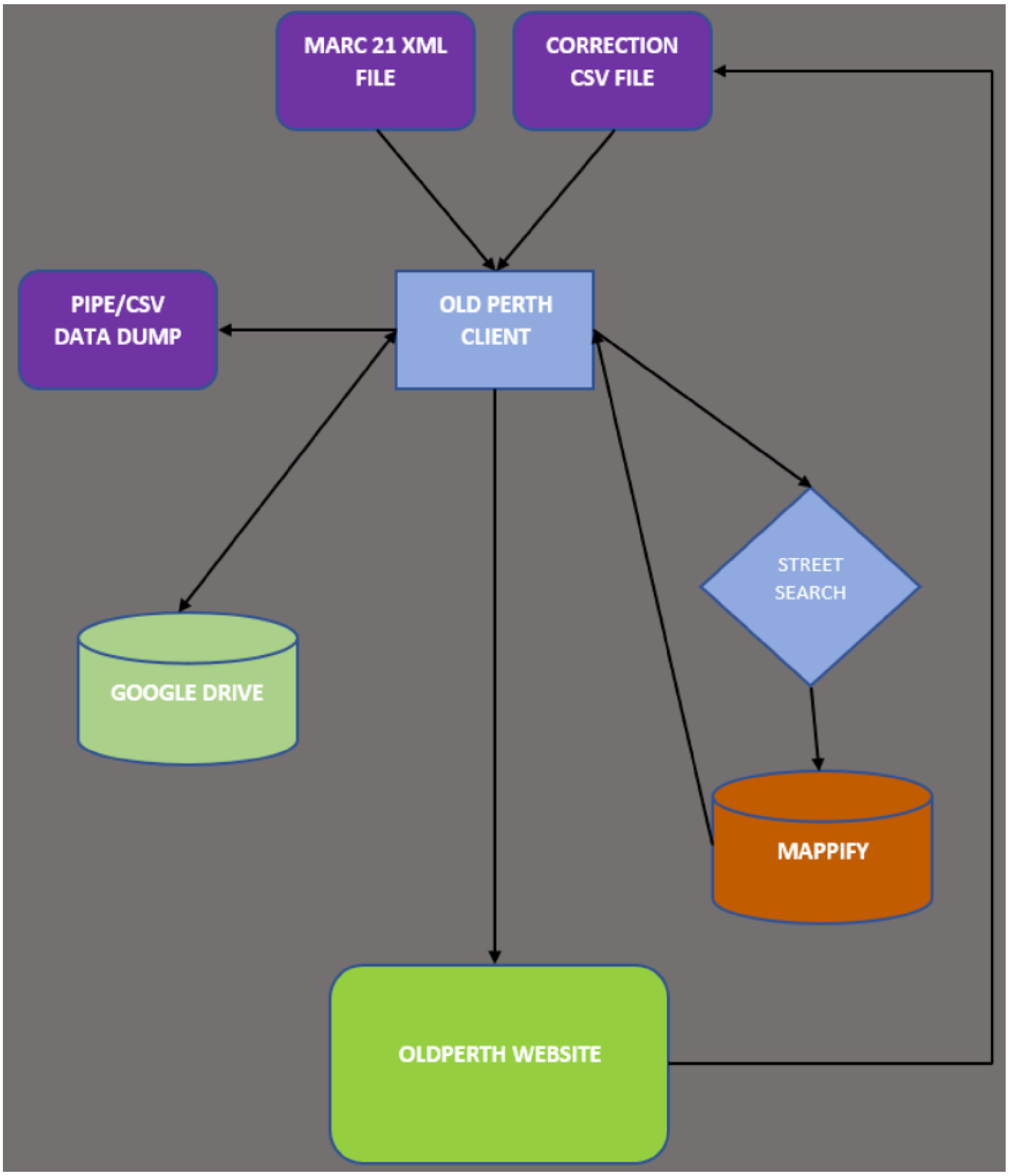


Figure 2. *OldPerth Website Logic* [6]

OldPerth - Current Infrastructure

The current iteration of OldPerth, a result of previous projects, comprises a repository of more than 12,000 geolocated images. The images are sourced from the state library database at www.purl.slwa.gov.au and are presented on the interactive map.

The website demonstrates robust functionality and effective performance; however, its suitability for kiosk applications intended for public spaces is confined. For a public console, important considerations includes privacy and security measures, interactive functionalities, automated content resetting and include other content formats. The current OldPerth site is not advisable for deployment as kiosk in public consoles due to the ensuing reasons:

The website includes numerous external links to enhance its functionalities. These links extend beyond static links of social media and branding links. Each image is linked to the SLWA catalogue. The presence of many external links raises concerns about unauthorised redirection, requiring a much robust strategy. An alternative method to embed links to site is needed to provide continue access to the library via consoles without altering the content or removing functionalities of OldPerth website.

Currently, the website remains an idle map unless a user actively interacts and browses through the images. For a kiosk purpose, there is a need of an interact feature capable of capturing users' attentions. An automated feature operating in the background that reveals interesting images without user input. Through this, it enhances the likelihood of individuals exploring the kiosk and exploring the collections of images, thus bringing wider audience.

The website lacks an auto-reset functionality for resetting the page. The absence of an auto-reset mechanism presents challenges when a user leaves the map in an uninteresting location. Periodic resetting of the website is crucial to ensure that users, especially those encountering the site for the first time, are not confounded by outdated or random information. implementation of an auto-rest functionality becomes imperative to reset the map to a more engaging area, specifically the Perth CBD, where the majority (90%) of images are geolocated and displayed on the map.

3. Method

The project employs various technologies and programming languages. The frontend of OldPerth website is developed using HTML, CSS, jQuery, and JavaScript, while the backend is primarily developed in Python. Working with a previously developed project introduces numerous nuances; therefore, we have chosen to adhere to these languages and technologies throughout the course of this project.

QR Code Generation

The incorporation of QR code into the OldPerth kiosk version serves the purpose of replacing external links from the site. This still allows access to the content on the site but ensures the privacy and security of the site by restricting the users to explore further at their own convenience without redirection.

Implementation of QR Code

A web component [10] from bitjson [11] was utilised to implement the QR code feature on the site. We use combination of both Cascading style (CSS) and jQuery to build the design aspect of the QR while JavaScript was used to build the logic. A function “*generate*” that takes the parameter link, external link of the content, to generate the QR web component. Once the QR is generated, it is the appended to the <div></div> element with ID “*qrContainer*” declared in the HTML.

Integration of QR Code

To integrate QR code functionality into the OldPerth kiosk version, a “*generateQR*” function is created that takes the link parameter and subsequently pass it to the generate function. Additionally, it also handles to logic associated with opening and closing of the QR code.

Two considerations were taken into account during integration: the static <a> tags within the HTML and the dynamic image links sourced from www.purl.slwa.wa.gov.au. To satisfy both conditions, we utilised ECMAScript Module Syntax [12], enabling the incorporation of QR codes into dynamically linked images. For this, we modified the “viewer.js” JavaScript file to call the generateQR function instead of hyperlinking to the library catalogue. For the static <a> tags, a global assignment of the generateQR function was implemented and an “onClick” event listener is added to all <a></a> tags configured to execute the generateQR function.

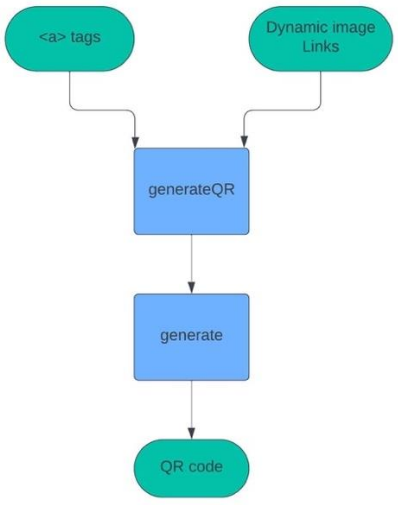


Figure 3. *QR code generation flowchart*

Interactive Feature & Auto Reset

The auto-reset functionality serves a dual purpose by showcasing interactive passive feature to capture visitors' attention while also guaranteeing a seamless user experience through the automatic resetting of the website under specific conditions. These primary conditions include the automatic display of images on the map during idle states, resetting the site in the event of temporary layout modifications, such as images being temporarily filtered, and initiating a site reset at five-minute intervals.

Fetch Random Location

The interactive functionality utilises the by-location directory, wherein images sharing identical geo-coordinates are grouped and presented on the map. An asynchronous getData function [13] was created to fetch data from the by-location directory on the server. The reason for using an asynchronous function is not to delay any other implementations when retrieving the by-location directory from the server. This asynchronous function uses the Fetch API [14] to retrieve HTML content from the server. Subsequently, we used regular expressions to extract JSON files from the HTML content. Once the files are obtained, a random JSON file from the by-location directory is chosen, and the IDs of images within that file is fetched using another API call.

Inactivity Detection

In order to detect user inactivity, it relies on a combination of “mousemove” and “onClick” event listeners. The *idleStateThreshold* constant sets the time threshold for considering the system as idle. For the current iteration, we have set the *idleStateThreshold* to be 30 seconds. If there is no mouse movement or click within that threshold, interactive feature is initialised through the function *initialiseInteractiveProcess*.

Automatic Interactive feature

Upon detecting user inactivity within the *idleStateThreshold* period, the "*buildInteractiveFeature*" function is invoked. Subsequently, when this function is called, the "*getData*" function is executed. The "*getData*" function, responsible for selecting a random location, retrieves all the IDs of the images associated with the chosen location. These image IDs are then passed into the "*loadImage*" function, which assesses whether there is only one image at that location or multiple images.

In the case of a single image, the "*loadSingleImage*" function is invoked. Alternatively, if there are multiple images, the "*loadMultipleImages*" function is invoked. The "*loadMultipleImages*" function iterates through the images up to maximum of 5 images and transition to the next location. The execution of the "*buildInteractiveFeature*" function continues until a user engages with the site.

Stop Automatic Interactive feature

If a user clicks on the site or click on the designated element with ID “#exit-slideshow”, it immediately stops the building interactive process by calling the “*stopInteractiveFeature”* function, allowing the user to interact with the site in a normal manner. The function clears all timeouts and prevents the auto-reset from proceeding.

Layout Change

The site has numerous features that could affect the layout of the site and make the site uninteresting for the subsequent users by someone. To ensure that layout change is reset, an event listener function using jQuery is created to monitor changes in a filter setting, represented by the #time-slider element. A combination of “*mousemove*” and slider events are used to detected the change in the layout. When the filter is changed, a variable “*sliderChanged*” is set to true. If there is no subsequent user activity then “*userActiveAfterFilter*” is set to false and the website is reset. This mechanism ensures that the website is only reset when the user intentionally changes the filter setting and there is no subsequent user activity

Reset Every Five Minutes

A "*mousemove*" event listener is employed to monitor user activity. Upon detection of a "*mousemove*" event, the Boolean variable "*userActive*" is assigned a value of true; otherwise, it is set to False. The periodic detection of "*mousemove*" events occurs at intervals of 5 minutes, as defined by the variable "*autoResetInterval*" within the setInterval asynchronous function. In the absence of user activity subsequent to the setting of "*userActive*" to False, the site undergoes a resetting process.

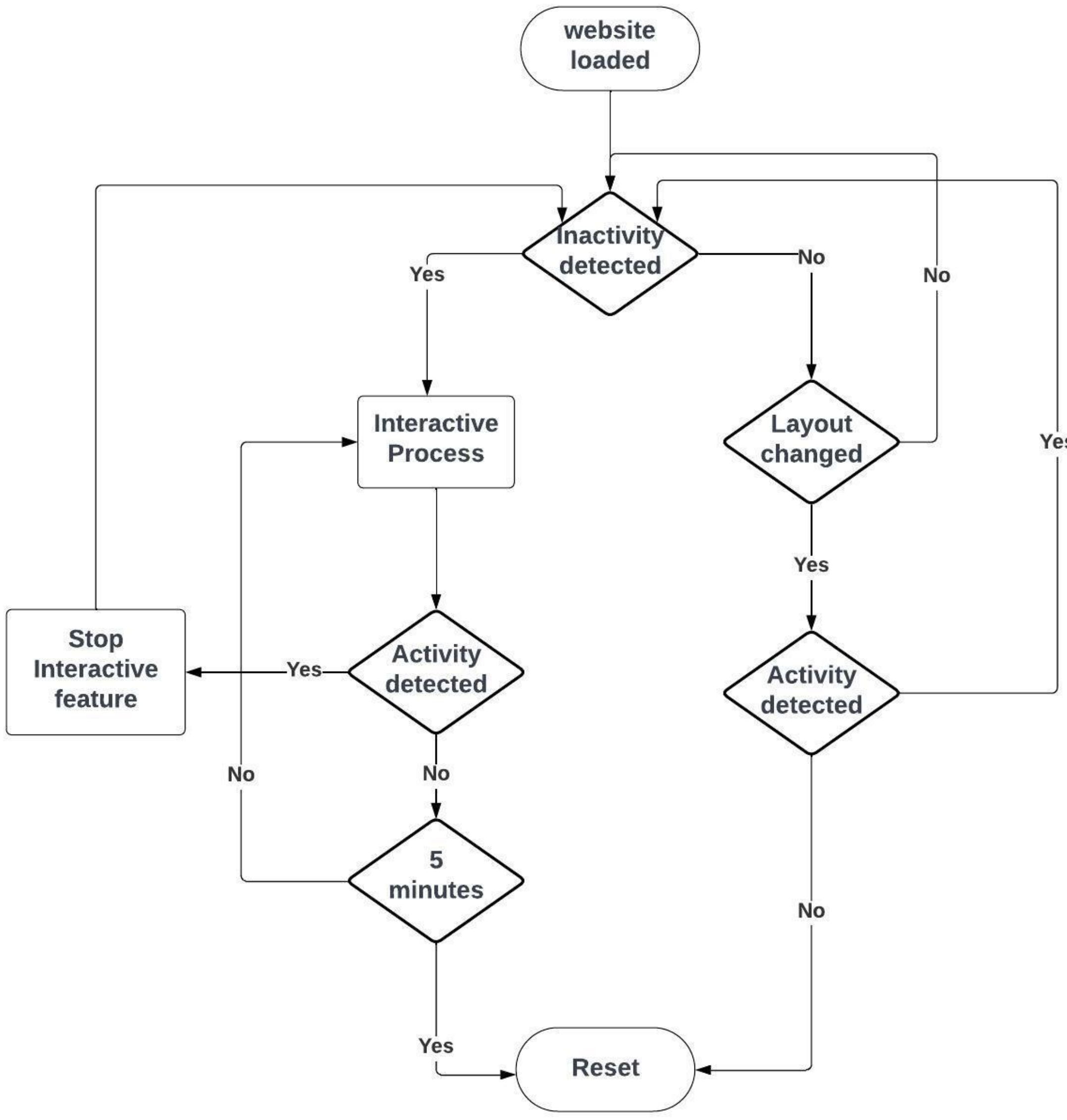


Figure 4. *Auto-reset logic*

Video and Audio Prototype

The State Library's catalogue collection is not limited to photographs but also includes collections of other formats, such as audio and video files. The incorporation of audio and video content further enhances the richness of OldPerth. To integrate these file formats into the site, the initial step in the process was to parse the MARC 21-formatted audio and video catalogue data.

The existing functions of OldPerth's backend could not be applied to read the audio and video MARC records due to structural differences. As a result, we modified the 'old_perth.py' file. A new variable, "file_type," is created to specify the type of file to be read. Once the file type has been identified, the process of reading the audio and video catalogue resembles reading the image catalogue data.

The existing data structure of OldPerth includes three fields from the metadata: author, year, and comment. However, audio and video MARC 21 formatted files lack a comment field "856"; instead, they contain a different field, "962," which holds relevant information. Despite these structural differences, the same data structure is applied to both audio and video files to avoid the need for complete redevelopment.

To address this difference, a new function "getAudioVideoData" has been created. Similar to the "getRecordData" function, which extracts these three fields from the image catalogue, "getAudioVideoData" extracts the following fields from the audio and video catalogue: "100" for author, "260" or "264" for the year, and the comment field "962."

Feature Changes

We updated couple of features to ensure information are relevant and update to date.

Branding Updates

- A "Donate an image button" has been removed instead replaced with SLWA log that has "donate an image button" as part of the logo
- The transparent SLWA Government Crest logo in the expanded show has been replaced by a newer version provided by SLWA

Language Changes

We made couple in language to enhance the clarity and efficiency of the language on the site. The phrase "Is this image in the wrong place?" previously appearing in the show expanded image display has been substituted with the term "Error" to provide a more direct indication of the issue. Additionally, the prompt "Please lodge an image correction" has been replaced with the more straightforward and action-oriented instruction "Help Fix It."

Fixed Search Bar

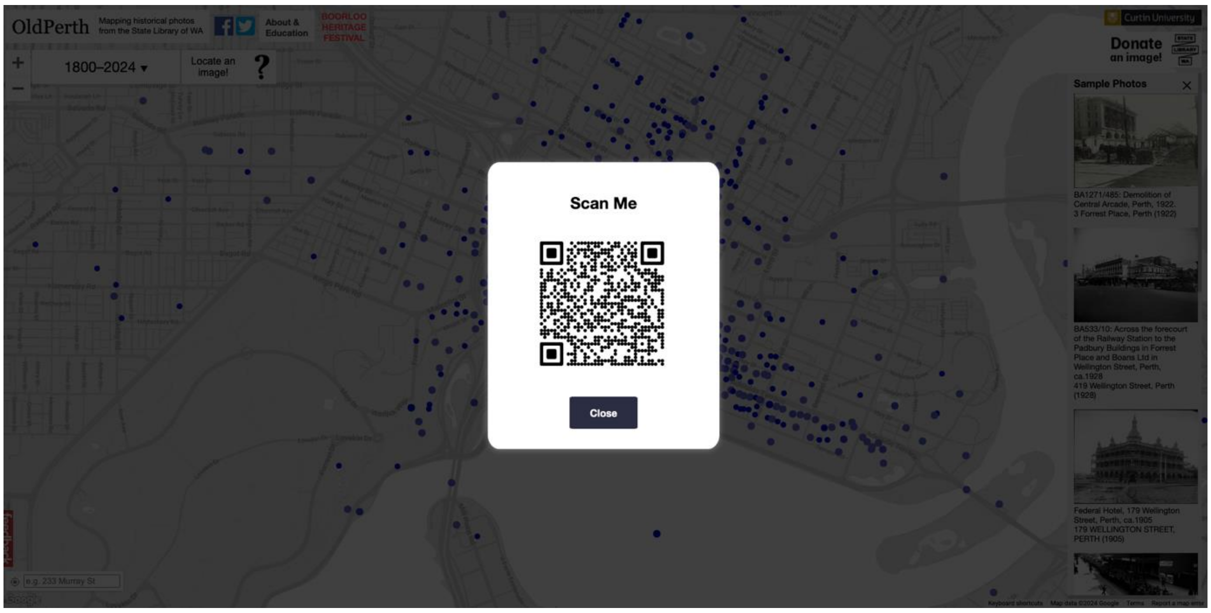
The search feature on the OldPerth was not functional due to restriction on the Google Maps API. To fix this, we modified the search.js JavaScript file to include new API key that enables to search feature without any problem.

4. Results

Completion of Goal 1: Replace Embedded Links with QR Code

Replace embedded links on OldPerth website with QR codes to enhance security and privacy.

The implementation of QR codes in the OldPerth kiosk version achieved the goal of eliminating external links from the site. We replaced 17 static links and a procedurally-generated QR code for 10000+ images that are linked to the SLWA's catalogue. When a user clicks on a link or an image, a QR code now dynamically appears instead of opening a new tab. This ensures that all user activities, including browsing and navigation, remain within the site, preventing unauthorised use of kiosk. Furthermore, this approach aligns with contemporary digital trends particularly with kiosk consoles, where QR codes have become a widely accepted and user-friendly method for accessing information.

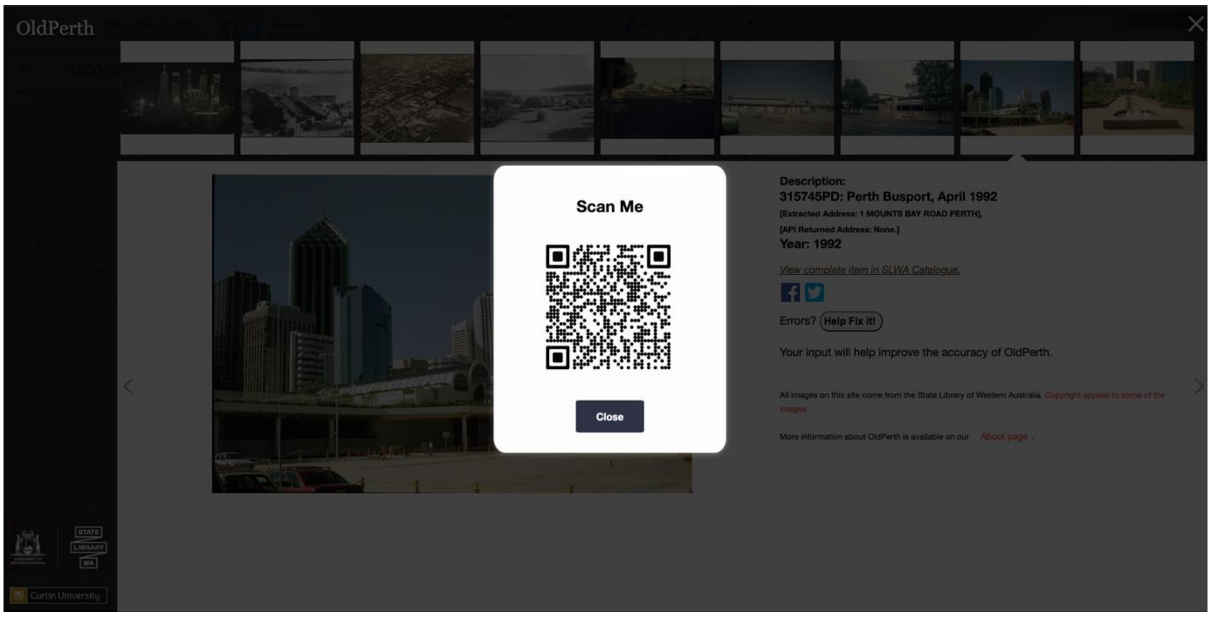


BA1271/485: Demolition of Central Arcade, Perth, 1922
3 Forrest Place, Perth (1922)

BA533/10: Across the forecourt of the Railway Station to the Padbury Buildings in Forrest Place and Boans Ltd in Wellington Street, Perth, ca.1928
419 Wellington Street, Perth (1928)

Federal Hotel, 179 Wellington Street, Perth, ca.1905
179 WELLINGTON STREET, PERTH (1905)

Figure 5. *QR code generated for social link*



Description:
315745PD: Perth Busport, April 1992
[Extracted Address: 1 MOUNTS BAY ROAD PERTH],
[API Returned Address: None.]
Year: 1992

View complete item in SLWA Catalogue.

Errors? Help Fix it!

Your input will help improve the accuracy of OldPerth.

All images on this site come from the State Library of Western Australia. Copyright applies to some of the images

More information about OldPerth is available on our About page .

Figure 6. *Dynamically generated QR Code*

Complete of Goal 2: Develop an passive interactive feature and auto-reset mechanism

Develop and integrate an interactive feature on the website to autonomously select a location, browse images, and reset website after a set period, enhancing visitor engagement and interaction.

In instances of user inactivity exceeding 30 seconds following the initial loading, the website transitions from displaying an idle map to showcasing random images from various locations on the map. If only one image is available at a location, it is shown for a duration of 10 seconds. However, if multiple images are present, it iterates through each image, limited to a maximum of five, with each image being displayed for 10 seconds. This limitation is in place to ensure the system loads multiple locations, presenting a more diverse range of images rather than adhering to a single location. Upon the loading of images, the system seamlessly transitions to automatically select the next random location, and this automated loading and transitioning process continues in the absence of user activity.

An automatic reset takes place if there is a layout change on the website and no user activity is detected within 10 seconds of the layout being altered. Furthermore, the website resets every 5 minutes in the absence of user activity, ensuring the system remains dynamic and responsive.

Completion of Goal 3 : Include audios and videos

Investigate methods for incorporating other formats of heritage collections into the website, with a specific focus on audio-visual content such as films or oral histories.

Built a prototype of including other content formats onto the website. Currently, 3 audios and videos have been integrated to the site. These audios and videos are manually geolocated through OldPerth's geolocation process and then integrated to the OldPerth's database.

Complete of Goal 4: Update the site

Review codebase of the site and update the site with the latest branding and relevant information.

Key changes were made to both OldPerth website and OldPerth Kiosk version.

1. Fixed the search feature by updating the Google Maps API
2. Changed 'ga' to 'gtag' in Google Analytics in adherence to Google's updates.
3. Updated the State Library's logo and refined the "Donate an Image" button.
4. Refined language for conciseness and clarity.
5. Conducted general codebase refactoring for stability, updating dependencies, and integrating new features.

These changes enhance site stability, align with the latest branding standards, and display relevant information on the site.

5. Discussion

QR Code Performance

The Bitjson/qr-code [11] generator was used for QR code generation due to its easy to use and lack of external dependencies. It is highly customisable and renders SVG-based HTML element, facilitating easier integration into our site.

For the current use case, the QR code generated automatically closes after 20 seconds if left unattended. This is to ensure that uninterrupted execution of other features without compromising visualisations or the overall user experience. The duration can also be changed according to the usages, but for this purpose, we deemed 20s would be an appropriate time for people to take their device and scan the QR code.

Automatic Interactive Feature

During the initial phase of developing the passive interactive feature, the intention was to select a random image and display it for a duration of 10 seconds before proceeding to showcase the next random image. This intended implementation utilised a random function to generate a number within the range of 0 to 12000, using the generated number as an ID to fetch the corresponding image. However, this approach proved unfeasible primarily due to having few images without geolocation data and others being disabled. Consequently, an alternative strategy was adopted, whereby a random location is chosen, and images associated with that location are displayed.

In the current passive interactive implementation, the selection of a random location is executed by retrieving the "by-location" directory through an HTTP server. Within the "by-location" directory, it contains JSON files with array of images. To get a random json file, a JavaScript file "auto-reset.js" created that uses combination of JavaScript Promises and regular expressions to fetch a random JSON file. However, an alternative approach could involve utilising the Node.js file system to remotely access the "by-location" directory for the random selection of a location. In this scenario, the "auto-reset.js" file could be bundled, similar to other JavaScript files, using Webpack [15] to enhance loading speed.

Through the implementation of these two features, the site has become more robust and secure for use in a public console for a kiosk. More importantly, with the automatic passive loading of images, we aim to capture users' interest and draw their attention to the remarkable SLWA's catalogue.

Audio and Video Inclusion

This was just an initial prototype demonstrating how various content formats could be integrated into the site. Currently, the geolocation process is carried out manually. This is because audios and videos have different data structures that make them more challenging to geolocate compared to images. This difficulty is compounded by the dynamic nature of audios and videos.

Additionally, audios and videos do not currently offer a playback option. The main purpose of incorporating audios and videos is to make them playable on the kiosk and engage more users. However, to enable this functionality, a complete redevelopment of the backend and modifications to some features of the frontend would be required.

Future Work

There are many areas in which OldPerth Kiosk version could be optimised for use in public consoles. Some of the notable work could be:

1. Develop a fully offline version of the kiosk. This would include developing comprehensive image catalogue caching or locally stored and using Map API that supports offline usage.
2. Enhance the program's capabilities to include geolocation of audio-visual content, broadening the scope of available media types.
3. Integrate spatial movement features for notable historical figures associated with OldPerth
4. Implement version control mechanisms to manage two distinct sites, ensuring both have seamless access to shared resources.
5. Revise the language used in the about page and replace all hyperlinks with QR codes.
6. Establish a connection between the backend and frontend systems. Currently, manual updates are required for files modified by the backend to reflect in the frontend folder. Consider implementing an automatic connection through a database, enabling the frontend to retrieve data seamlessly from the database.
7. Conduct thorough testing post-kiosk deployment and evaluate the user engagement and interactions. Use the insights to improve the kiosk experience.

6. Conclusion

The primary objective of the OldPerth project was to build a self-contained, user centric and interactive version of the currently existing OldPerth website for individuals physically visiting the State Library. We were able to achieve our goals of developing this kiosk version through integration of QR codes into the site instead of hyperlinks, development passive interactive feature and developing a prototype to include alternative content formats, particularly audio-visual forms.

Through the implement of QR codes, the kiosk version enhances security by restricting users from navigating to unauthorised sites. 17 static hyperlinks and more than 10000 dynamic image hyperlinks were replaced with QR codes. QR codes provides the same functionality as would the hyperlinks that allows users to dive deeper into specific photograph or explore the State Library's catalogue at their own convenience.

The passive interactive feature introduces a dynamic aspect by randomly selecting locations and showcasing images, effectively preventing the display of an idle map. Furthermore, in instances where users significantly change the map layout or place the map on an unintended positions on the map, the auto-reset functionality ensures the website resets, providing a seamless and engaging experience for subsequent users. Integration of this feature is anticipated to foster increased user interaction and engagement with SLWA's extensive photographic collections and the OldPerth website, thereby reaching a broader audience, including researchers and individuals retracing memories through the lanes of the past

While the project has made significant strides and ready to be deployed, the ultimate test lies in deploying the website on public consoles and testing its performance. We looking forward to testing out website on consoles and gathering valuable insights through the tests, that allow to enhance user engagement and interaction. Through this, we aim to ensure that the kiosk version not only meets but surpasses its objectives, establishing itself as a user-friendly and impactful resource to wider audience.

7. Acknowledgements

I would like to express my gratitude to the following people and organisation for the immense support during the project:

- My supervisors Dr. Andrew Woods (Curtin HIVE), Barbara Parnaby (State Library of Western Australia), Debra Jones (State Library of Western Australia), and Michael Ovens (Curtin HIVE).
- Ash Doshi and other HIVE staffs for all the technical assistance.
- The State Library of Western Australia for the financial support that made this project possible.

8. References

[1] Library of Congress, “Bibliographic,” Library of Congress, December 2023. [Online]. Available: https://www.loc.gov/marc/bibliographic/. [Accessed 10 January 2024].

[2] M. Collins, “OldPerth - Mapping Historial Photographs of Perth,” Curtin HIVE , Perth , 2020.

[3] M. Shelton, “ThickShake : Towards a Smarter Library Catalogue,” Curtin HIVE , Perth, 2019.

[4] P. Labs, “Mappify,” 2016. [Online]. Available: https://mappify.io/.

[5] D. E.King, “Dlib-ml: A Machine Learning Toolkit,” Journal of Machine Learning Research, p. 4, 2009.

[6] G. Bradski, “The OpenCV Library,” Dr. Dobb's Journal of Software Tools, 2000.

[7] R. Smith, “An Overview of the Tesseract OCR Engine,” in Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR), 2007, pp. 629-633.

[8] F. Gabriel , M. Mark and S. Ed, “Pymarc,” 2005. [Online]. Available: https://github.com/edsu/pymarc.

[9] Python Core Team, “Python: A dynamic, open source programming. Python Software Foundation,” 2001. [Online]. Available: https://www.python.org/.

[10] Mozilla Corporation, “Web Components,” 1988-2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_components.

[11] J. Dreyzehner, “bitjson/qr-code,” 27 February 2023. [Online]. Available: https://qr.bitjson.com/.

[12] ECMA International, “publications and standards,” June 2023. [Online]. Available: https://ecma-international.org/publications-and-standards/standards/ecma-262/.

[13] Mozilla Corporation, “async function,” 1998-2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function.

[14] Mozilla Corporation, “Fetch API,” 1998-2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.

[15] OpenJS Foundation, 19 February 2014. [Online]. Available: https://webpack.js.org/.

9. Appendices

Appendix 1 – Modifying QR Code Generation (qr-code.js)

1. Navigate to the folder “js/qr-code.js”.
2. Make the desired changes
3. Bundle using command “npm run build”
4. Run “npx http-server”

The changes to QR code generation will be reflected.

```
.on("click", ".og-fullimg > img", function () {
               var photo_id = $("#grid-
container").expandableGrid("selectedId");

               generateQR(libraryUrlForPhotoId(photo_id));
});
```
Code 1: *Dynamic generation of QR code*

Appendix 2 – Modifying Automatic Feature and Resetting (auto-reset.js)

1. Navigate to the folder “js/bundle/auto-reset.js”.
2. Make the desired changes
3. Run “npx http-server”

The changes to QR code generation will be reflected and no bundling is required.

Appendix 3 – Modification to old_perth.py

```
def marc_read():
    print('\nPlease ensure the file is located in 
dist\LibClean\marc_data')

    while True:
        file_type = input("What type of file you would like to 
read. Type i for images or av for audio-video: ")

        if file_type.lower() in ('i', 'av'):
            break
        else:
            print('Invalid selection. Please enter i or av.')

    f_name = input('\nName of the file:')

    valid = False
    while not valid:
        num = input('Number of records to read (hit enter for 
all records):')
        if num == '':
            num = None
            valid = True
        elif is_int(num):
            num = int(num)
            valid = True
        else:
            print('Invalid selection')

    try:
        frame.readxml(f_name, file_type, num)
    except Exception as e:
        print(f"Error reading file: {type(e).__name__} - {e}")
```

Appendix 4 – Addition of getAutoVideoData to Read Audio & Video MARC files

```
def getAudioVideoData(self, record):
        # Extract information from this
        # Subfield 100 for author if there
        # Subfield 262
        # Subfield 962 for comment and url
        images = []
        yrpat = re.compile(r'(\d{4})')
        if "100" in record and record["100"]:
            author = record["100"]["a"]
        else:
            author = None


        if "260" in record and record['260']:
#Extracts year from entry
            year = re.findall(yrpat, record['260'].value())
            if len(year) > 0:
                year = int(year[0])
            else:
                year = None
        elif "264"in record and  record['264']:
            year = re.findall(yrpat, record['264'].value())
            if len(year) > 0:
                year = int(year[0])
            else:
                year = None
        else:
            year = None

        if "962" in record and record["962"]:
            comments = [val for val in
record["962"].get_subfields("t")]
            images.append(comments)


        # Check if "962" field exists before accessing its
subfields
        if "962" in record and record["962"]:
            urls = [val for val in
record["962"].get_subfields("u")]
            images.append(urls)

        return author, year, images
```