High-Level Technology Choices for Development of the Project Omega - Minimum Viable Project

Author: Adam Retter (Evolved Binary)
Date: 2022-08-26

Background

During Phase 1 of Project Omega in 2020, a task for identification and selection of technologies for the purposes of developing an implementation of Project Omega was undertaken. The conclusion of this task was the selection of a Cloud environment, and two Programming Environments for development and implementation of the complete system: Amazon Web Services for the Infrastructure Services, TypeScript for development of in-Web-Browser functionality, and Scala for development of middleware, backend, and database services.

- Amazon Web Services was chosen for the Infrastructure Services as it provides a Cloud Environment that is widely used across The National Archives. Amazon Web Services is able to provide all of the services that are envisaged to be required as components of Project Omega.

- TypeScript was chosen as it provides a modern, popular, and more robust development experience than JavaScript whilst compiling to JavaScript; which is the only technology option available for programming the Web Browser.

- Scala was chosen as it was identified that within The National Archives, there were two existing pools of development expertise: .Net within the Discovery team (who were in the process of transitioning to Python), and Scala within the Digital Archiving teams. It was recognised both that, The National Archives already had a large investment in the Java Virtual Machine (due to an array of existing commercial and bespoke Java and Scala applications), and that as .Net was on the way out, and Python had yet to become established (within The National Archives), the Digital Archiving teams would be able to better support the Omega application if needed.

- In addition Pentaho Data Integration (Open Source), a Java product, was selected for Extract-Transform-Load data conversion activities. Such a tool will be required for conversion of The National Archives existing Catalogue Data. It was selected as it may be accessed and extended either from Java or Scala, and fit well with the choice of the Java Virtual Machine for backed processing.

Recently (Q2-Q3 2022), there have been questions raised over the continued viability of using Scala at The National Archives. The main point of concern arises from The National Archives' difficulty in recruiting Scala developers (or any developers in general) who are perceived as demanding premium remuneration. It is believed the underlying cause of this recruitment issue is caused by two factors: (1) The National Archives' location in the advanced London market where developers are well rewarded, (2) the difficulty for The National Archives to offer a competitive package of remuneration for a position within the Public Sector.

In light of these questions, we have chosen to examine and re-evaluate our high-level technology choices for development of the Project Omega MVP.

Current Development Landscape

At the time of writing this report (Q2/Q3 2022), we have surveyed the current landscape of development technologies within The National Archives. This was achieved by both formal and informal discussions with Technical Architects and Technical Implementers across various projects within the Digital Directorate at The National Archives.

There remains a legacy investment in .Net technologies within the Digital Services (Discovery) Team, although much of this will eventually be eliminated as the Discovery System is scheduled to be replaced with a new system named Etna.

Whilst Discovery utilised an on-premise infrastructure, it is in the process of being migrated into Amazon Web Services. Etna utilises Amazon Web Services, CIIM (Proprietary), and Open/ElasticSearch (Open Source) at its core, both Java products which match well with The National Archives existing investment in the Java Virtual Machine. Etna also utilises: Kong Gateway (Open Source), a Lua product, which is a new technology for The National Archives, and Wagtail CMS (Open Source), a Django (Open Source) and Python product which fits with their earlier goal of migrating from .Net to Python.

There is still a strong existing and ongoing investment in Java Virtual Machine technologies from within the Digital Archiving teams. We are seeing continued use of both Java and Scala in existing and new projects. Digital Archiving righly continues to maintain its on-premise infrastructure for Archival purposes, but has also introduced some use of Amazon Web Services for external customer facing solutions. The new Transformation Engine project has been investing into Talend Data Integration (OpenSource and Proprietary), a Java product, and development of bespoke components in Java. The Transfer of Digital Records team is utilising Amazon Web Services, Scala for development of a bespoke portal, and has re-introduced the Play Framework (Open Source), a Scala product that was previously deployed within the Digital Repository Infrastructure project.

Additionally in some newer projects such as Gradated Access within Digital Archiving, that require smaller web-portals, there are plans to deploy Django and Python as the middle/backend solution.

The Legislative projects have continued their investment in Amazon Web Services, MarkLogic (Proprietary) which utilises XQuery and XSLT for development, and Ontotext (Proprietary) which utilises SPARQL for development. New projects within Legislation have also introduced new technologies around Natural Language Processing, and web-portals utilising Django and Python.

In summary, there appear to be three strong factors at play in the current development landscape within The National Archives:

1. An expanding investment in Amazon Web Services, driven by a migration from on-premise infrastructure to the cloud.

2. A continued investment in the Java Virtual Machine, and use of both Java and Scala. Of some surprise here was the additional adoption of Java by the Etna project who previously held the goal of transitioning from .Net to Python.

3. An emerging investment in Django and Python from Etna, and although limited to smaller services, also an investment in this area from both Digital Archiving and Legislative projects.

Project Omega Requirements

The core of Project Omega is a graph data store which will hold the canonical information of all assets of The National Archives. The goal is for this information base to grow and be enriched over time as more information from across the organisation is subsumed into it.

Simply placing all of the assets information into a large graph is an excellent start, but it is not particularly useful if it cannot be utilised and updated across the business. As such the Technical Architecture for Project Omega demands a layered approach. In many applications APIs to access and update data are provided as either an after-thought or a bolt-on of the applications Web User Interface. In Omega, the API is designed from the start as a first class citizen to be utilised by any business unit or member of staff (assuming appropriate access rights). In addition the Web User Interface in Omega is built atop this API, allowing us to validate our API design and demonstrate how to build applications atop the Omega API.

Therefore, the technical Systems Architecture for Project Omega takes a layered approach, with each layer building on and communicating with the previous. The four layers, from back-end to front-end, are:

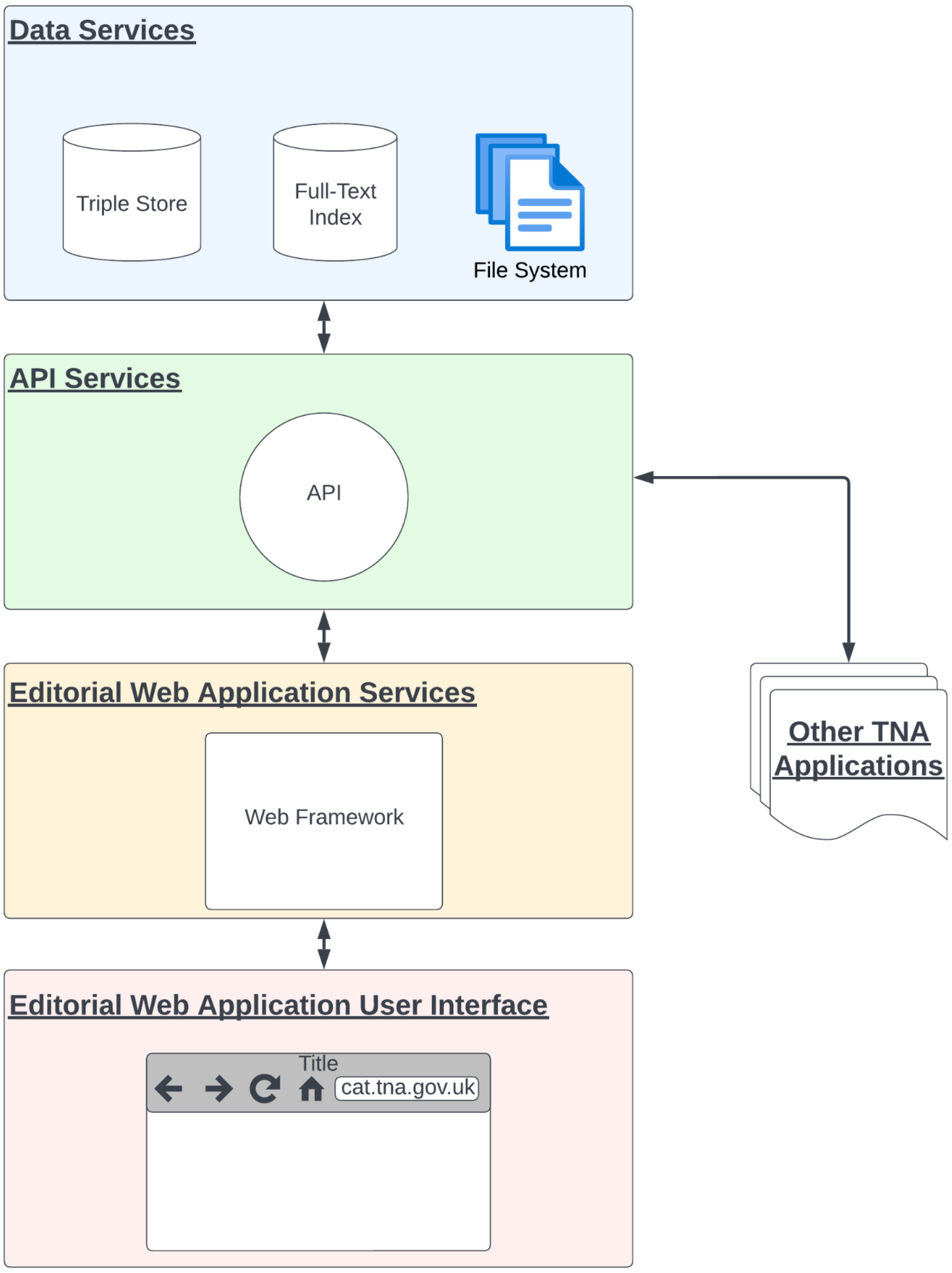
1. **Data Services**
   This layer contains the data storage, indexing, and querying capabilities. It is envisaged that this will be made up of a Triple Store for holding the Omega Graph(s),

a Full-Text Index, and FileSystem storage.

2. **API Services**
   This layer contains all APIs that deliver and receive data to and from both, the Editorial Web Application Services, and other applications within The National Archives. This layer is designed to be resilient and decouple any National Archives applications from the Data Services.

3. **Editorial Web Application Services**
   This layer provides the communication between the Web Application User Interface and the API Services. For the purposes of meeting a GOV UK goal for User Interfaces to be built from a position of Progressive Enhancement, it is therefore also responsible for the Service Side rendering of any dynamic aspects of the Web Application User Interface.

4. **Editorial Web Application User Interface**
   This is the layer that the Editorial end-users interact with. It executes within a Web-Browser and communicates with the Editorial Web Application Services.


Data Services
Triple Store
Full-Text Index
File System
API Services
API
Editorial Web Application Services
Web Framework
Editorial Web Application User Interface
Title
cat.tna.gov.uk
Other TNA Applications

Options for Project Omega Technology Choices

When considering changing the technology choices for Project Omega, we have re-examined our previous technology choices, considered those now in use within other projects at The National Archives, and examined a variety of technologies not currently in-use at The National Archives.

In general we consider our original choice of Amazon Web Services for providing the required Infrastructure Services as the correct one. Amazon Web Services use within The National Archives has been expanding, and is still likely able to provide all of the required services envisaged for Project Omega and scale up as needed. Herein we further examine the technology options for each of the four layers of the system.

1. **Data Services**
   We have examined several possible Triple Store implementations:
   - **Apache Jena TDB**
     Licence: Open Source - Apache 2.0
     Latest Release: 4.6.1 (2022-09-01)
     Language: Java

   - **BlazeGraph / Neptune**
     Licence: Open Source - GPL 2, or AWS Proprietary SaaS offering called Neptune
     Latest Release: 2.1.6 RC (2020-02-03)
     Language: Java

   - **Ontotext GraphDB**
     Licence: Proprietary
     Latest Release: 10.0.2 (2022-08-01)
     Language: Java

   - **RDF4j**
     Licence: Open Source - BSD 3-Clause
     Latest Release: 4.1.1 (2022-09-02)
     Language: Java

   - **Virtuoso**
     Licence: Open Source - GPL 2, and Proprietary options
     Latest Release: 7.2.7 (2022-05-18)
     Language: C

   We have examined several possible Full-Text Index implementations:
   - **Apache Lucene**
     Licence: Open Source - Apache 2
     Latest Release: 9.3.0 (2022-07-28)
     Language: Java

- **Elastic Search / Open Search**
  Licence: Open Source and Proprietary, or SaSS offerings e.g. Elastic Cloud / AWS Open Search
  Latest Release: Elastic Search 8.4.1 (2022-08-30) / Open Search 2.2.1 (2022-09-01)
  Language: Java

- **Sphinx**
  Licence: Open Source - GPL 2, and Proprietary
  Latest Release: 3.3.1 (2022-07-06)
  Language: C++

- **Xapian**
  Licence: Open Source - GPL 2
  Latest Release: 1.4.20 (2022-07-04)
  Language: C++

2. **API Services**
   As this layer is required to be resilient and decouple any National Archives applications from the Data Services, we have examined Message Oriented Middleware approaches. Building this layer utilising a Message Oriented Middleware approach requires two main components:

   - A Message Broker that can provide durable and resilient Message Queues and Topics, and routing of messages.

   - Business API Services that provide Catalogue Services by sending and receiving messages via the Message Broker to both the Editorial Web Application Services and other National Archives' applications.

   The Business API services will require bespoke development, whilst the Message Broker will be a 3rd-party component or service.

   We have examined several possible Message Broker implementations:

   - **Apache ActiveMQ**
     Licence: Open Source - Apache 2.0
     Latest Release: 5.17.2 (2022-08-25)
     Language: Java

   - **Apache Kafka / Amazon Managed Streaming for Apache Kafka (MSK)**
     Licence: Open Source - Apache 2.0, or AWS Proprietary SaSS offering called MSK
     Latest Release: 3.2.1 (2022-07-29)
     Language: Java

   - **Amazon Simple Queue Service**
     Licence: AWS Proprietary SaSS offering

Latest Release: Unknown
Language: Unknown

- **RabbitMQ**
  Licence: Open Source - Mozilla Public Licence
  Latest Release: 3.10.7 (2022-08-02)
  Language: Erlang

3. **Editorial Web Application Services**
   This layer must provide the logic for the Editorial Web Application User Interface. It is responsible for any computation that is required to change the state of pages/screens within the User Interface. It is also responsible for providing the communication between the User Interface and the API Services, and any necessary reformatting or reframing of data.

   Whilst the Editorial Web Application Services will require bespoke development, we have identified and examined several frameworks which could provide key facilities and therefore reduce development time. These frameworks provide typical features such as web serving, URI routing, authentication, etc., but each falls into one of two classes. The first class provides a toolkit of UI components which are coded server side and delivered to the User Interface, and the second class which provides no server side UI components and rather expects these to be completely provided by the User Interface layer.

   We have examined several possible toolkits with UI Components:
   - **Gatsby.js**
     Licence: Open Source - MIT
     Latest Release: 4.22 (2022-08-30)
     Language: JavaScript

   - **Gwt**
     Licence: Open Source - Apache 2.0
     Latest Release: 2.10.0 (2022-06-09)
     Language: Java

   - **Primefaces**
     Licence: Open Source - MIT
     Latest Release: 12.0.0-RC3 (2022-08-24)
     Language: Java

   - **PrimeReact**
     Licence: Open Source - MIT
     Latest Release: 8.4.0 (2022-08-22)
     Language: JavaScript

   - **Vaadin**
     Licence: Open Source - Apache 2.0
     Latest Release: 23.2.0.rc1 (2022-09-32)

Language: Java / TypeScript

- **Wt**
  Licence: Open Source - GPL 2 with OpenSSL Exception
  Latest Release: 4.8.0 (2022-07-08)
  Language: C++

We have also examined several possible toolkits without UI Components:

- **Blade**
  Licence: Open Source - Apache 2.0
  Latest Release: 2.1.2.RELEASE (2022-05-09)
  Language: Java

- **Django**
  Licence: Open Source - BSD 3-Clause
  Latest Release: 4.1 (2022-08-03)
  Language: Python

- **Express.js**
  Licence: Open Source - MIT
  Latest Release: 4.18.1 (2022-04-29)
  Language: JavaScript

- **Play**
  Licence: Open Source - Apache 2.0
  Latest Release: 2.8.16 (2022-06-02)
  Language: Java / Scala

- **React.js**
  Licence: Open Source - MIT
  Latest Release: 18.2.0 (2022-06-14)
  Language: JavaScript

- **Scalatra**
  Licence: Open Source - BSD 2-Clause
  Latest Release: 2.8.1 (2021-09-25)
  Language: Scala

- **Spring Framework**
  Licence: Open Source - Apache 2.0
  Latest Release: 5.3.22 (2022-07-14)
  Language: Java

- **Quarkus**
  Licence: Open Source - Apache 2.0
  Latest Release: 2.12.0 (2022-08-31)
  Language: Java

- **Vert.x**
  Licence: Open Source - Eclipse Public License 2.0
  Latest Release: 4.2.2 (2022-09-09)
  Language: Java / Groovy / Kotlin

4. **Editorial Web Application User Interface**
   This layer operates on the client-side and within the Web-Browser. The Web Browser tightly constrains our available technology options for building a User Interface, the base languages available are therefore limited to HTML, CSS, and JavaScript.

   We have examined several possible client-side toolkits of UI Components:
   - **Angular Material**
     Licence: Open Source - MIT
     Latest Release: 14.2.0 (2022-08-25)
     Language: HTML, CSS, and JavaScript

   - **BaseWeb**
     Licence: Open Source - MIT
     Latest Release: 12.1.2 (2022-09-02)
     Language: HTML, CSS, and JavaScript / TypeScript

   - **Bootstrap**
     Licence: Open Source - MIT
     Latest Release: 5.2.0 (2022-07-19)
     Language: HTML, CSS, and JavaScript[1]

   - **Evergreen**
     Licence: Open Source - MIT
     Latest Release: 6.10.3 (2022-07-18)
     Language: HTML, CSS, and JavaScript

   - **GOV UK Frontend**
     Licence: Open Source - MIT
     Latest Release: 4.3.1 (2022-08-18)
     Language: HTML, CSS, and JavaScript

   - **HMRC Frontend**
     Licence: Open Source - Apache 2.0
     Latest Release: 5.5.0 (2022-08-31)
     Language: HTML, CSS, and JavaScript

   - **Ministry of Justice Frontend**
     Licence: Open Source - MIT
     Latest Release: 1.4.2 (2022-05-18)
     Language: HTML, CSS, and JavaScript

[1] Not all components require JavaScript.

- **MUI Core**
  Licence: Open Source - MIT
  Latest Release: 5.10.4 (2022-09-06)
  Language: HTML, CSS, and JavaScript / TypeScript

- **Transfer of Digital Records Frontend (TNA)**
  Licence: Open Source - MIT
  Latest Release: v1740 (2022-09-06)
  Language: Scala, HTML, CSS and TypeScript

- **The National Archives Frontend Toolkit**
  Licence: Unknown
  Latest Release: 0.9.1 (2020-12-10)
  Language: HTML, and CSS

- **Web Experience Toolkit (wet-boew)**
  Licence: Open Source - MIT
  Latest Release: 4.0.51.3 (2022-08-26)
  Language: HTML, CSS, and JavaScript

Recommendations for Project Omega Technology Choices

As the development team for Project Omega is greatly constrained by available resources (currently 1 Technical Architect, 1 Technical Lead, and 2 Software Engineers) within the development team, we first and foremost recommend keeping the number of different technologies used within the project to a minimum. This allows all members of the team to share skills, and to contribute interchangeably across the layers of the project.

Secondly, where possible we recommend the adoption of existing technologies that are already in-use within The National Archives. This allows support to be sought from other areas of the business as needed.

Thirdly, during this re-evaluation process we have examined many software applications, toolkits, and libraries spanning several different programming languages, by both identifying a suitable programming language already in use within The National Archives, and trying to keep the number of new technologies introduced to a minimum, we can eliminate many of the options available.

Whilst we could make almost any choice of programming language at 3 layers of our architecture, for our Web User Interface, the only realistic option for programmatically creating dynamic interactions within the Web Browser is to use the JavaScript programming language. Whilst we plan to follow a methodology of Progressive Enhancement and keep our use of JavaScript to a minimum, we are nonetheless likely forced into a situation where we will need some JavaScript within Project Omega.

If we must write some JavaScript, then we could logically extend our thinking to ask the question - “Why not build every layer in JavaScript?”. This would have the desirable property that there is only a single language in use within Project Omega. Equally, JavaScript is a skill which can be recruited for, ranking between number 1[2] and number 7[3] in the most popular languages, whilst required remuneration is lower[4] than that of Scala, or Python, and similar to that of Java.

Whilst this could be a great option in the near future for The National Archives to attempt to standardise on across the organisation, at present there is little or no experience with JavaScript at The National Archives (either within the Project Omega Team, or across the business), and there are no existing back-end systems written in JavaScript within The National Archives. We would strongly recommend The National Archives investigate and consider TypeScript as a standard language for future use, TypeScript solves many of the issues of JavaScript whilst still compiling to JavaScript for compatibility.

At present, The National Archives has existing investments in three key areas:

1. The Java Virtual Machine, i.e. Java and Scala
2. .Net - this is outgoing in favour of Python
3. Python - this is incoming in place of .Net

As Project Omega within The National Archives has an existing investment in Java and Scala, and the development team working on Project Omega is experienced in Java and Scala and has little Python experience, it would seem logical to continue with the investment in Java and Scala. Whilst it does appear that Scala developers can command a premium salary, we believe that The National Archives may be able to recruit existing experienced Java developers (or even apprentices) at a reasonable salary with the offer of upskilling them in Scala; Scala is generally perceived by the development community at large as a more desirable language to both learn and develop in.

Based on our recommendation of remaining with a Java and Scala stack, we look at each layer of the architecture below and justify the selection of further technologies at each layer.

Infrastructure

For providing the required Infrastructure Services we recommend the use of Amazon Web Services. We also recommend provisioning these using the Terraform (Infrastructure as Code), and Puppet (Configuration as Code) softwares. Amazon Web Services, Terraform, and Puppet, are already in use across The National Archives. Amazon Web Services appears to be their preferred Cloud Supplier to The National Archives and fits with GOV UK's Cloud First Policy[5].

We are aware that there is an effort to drive-down Cloud costs at The National Archives, as such we recommend carefully monitoring the situation and trying to isolate Project Omega from relying on specific Amazon Web Services in-case the need to migrate from Amazon Web Services arises.

[2] https://redmonk.com/sogrady/2022/03/28/language-rankings-1-22/
[3] https://www.tiobe.com/tiobe-index/
[4] https://survey.stackoverflow.co/2022/#technology-top-paying-technologies
[5] https://www.gov.uk/government/news/government-adopts-cloud-first-policy-for-public-sector-it

Data Services

We recommend the use of Amazon Neptune to provide an RDF Triple Store. Neptune fits well with the use of Amazon Web Services infrastructure, offers a SaSS option to reduce maintenance whilst the other options did not, and states impressive figures for scalability which will be required as Project Omega grows.

We recommend the use of Amazon Open Search to provide Full-Text Index services as this comes somewhat naturally due to Amazon's offering of a tight-integration between Neptune and Open Search.

We recommend the use of Amazon Simple Storage System (S3) to provide file-system storage. When operating within Amazon Web Services, unless you build your own storage system in EC2, then S3 is Amazon's SaaS offering and should suffice for the needs of Project Omega.

Where possible each of these services will be abstracted behind standard APIs to try and reduce vendor-lock-in. Neptune behind SPARQL Protocol, Open Search behind HTTP REST and JSON API, and S3 behind a HTTP REST and XML API.

Should we need to move away from Amazon Web Services cloud environment, it is also advantageous that there is an Open Source equivalent of Neptune - BlazeGraph, and Open Source versions of Open Search, each of these written in Java which fits well with our choice of development languages.

API Services

We recommend the use of Amazon Simple Queue Service to provide Message Broker services for the API. Simple Queue Service fits well with the use of Amazon Web Services infrastructure, and offers a SaSS option to reduce maintenance whilst the other options did not. It remains unclear at this time whether Amazon Simple Queue Service will offer sufficient functionality in the long term for Project Omega. Should more advanced functionality be required then Amazon Managed Streaming for Apache Kafka (MSK) should be explored.

The Message Broker service can easily be abstracted behind standard APIs and Protocols. We recommend the use of the JMS API and either the AMQP or STOMP protocols. In this manner should Simple Queue Service need to be replaced, or should we need to move away from Amazon Web Services cloud environment, then the Message Broker can be trivially replaced with a different implementation such as Apache ActiveMQ or Apache Kafka; also both Java products which fit well with our choice of development languages.

We recommend the use of bespoke Scala development for delivering the Business Services for the API. Our recommendation here is based on the existing investment that Project Omega have already made in developing a proof of concept Business Services API utilising

Scala. Such Business Services will communicate with the Message Broker using the JMS API and either the AMQP or STOMP protocols.

Editorial Web Application Services

Whilst the available toolkits at this layer that include UI components are impressive in their functionality and ease of use, we have discounted them. Those available toolkits, Gatsby.js, Gwt, Primefaces, PrimeReact, and Vaadin, all require large amounts of client-side JavaScript to be executed by the Web Browser to drive the User Interface. Such a need would conflict with our goal of a Progressive Enhancement[6] approach as set out of GOV UK, and also possibly lead to issues with Accessibility concerns (we are aiming for at least WCAG 2.1[7]).

The only such toolkit that supports Progressive Enhancement is Wt, however this requires development in the C++ language, which whilst known to some developers within Project Omega, is not widely used for Web Applications and is not used at The National Archives.

From the available tools that do not include UI components, Quarkus and Vert.x standout as modern Java frameworks, whilst Spring is well established in the Java arena, and Play remains the dominant choice in the Scala arena (whilst also providing a Java API).

As we are utilising Scala for the API Services, for the same reasons we also recommend the use of bespoke Scala development for delivering the Editorial Web Application Services. We additionally recommend the use of the Play framework to assist in the timely development and delivery of these services.
Both Scala and Play are already in use within other projects at The National Archives, such as The Transfer of Digital Records.

Editorial Web Application User Interface

Many of the toolkits that we evaluated for this layer have the same issue as the toolkits with UI components that we evaluated for the Editorial Web Application Services, that is to say that they require large amounts of client-side JavaScript and do not fit well our goals of Progressive Enhancement and Accessibility. For that reason alone, we have discounted Angular Material, BaseWeb, Bootstrap, Evergreen, and MUI Core as options.

Interestingly, all remaining toolkits that we identified at this stage are produced by either the UK Government, or by the Canadian Government. We note with some disappointment that there appear to be no other Web UI Toolkits available, excluding Wt, that consider Progressive Enhancement and Accessibility to be first class concerns.

We have learnt a great deal from studying the HMRC, and Ministry of Justice Frontends, each of these extends the GOV UK Frontend toolkit, but does so in a different manner and

[6] https://www.gov.uk/service-manual/technology/using-progressive-enhancement
[7] https://www.w3.org/TR/WCAG21/

utilises different technologies. HMRC Frontend makes use of Scala, whilst the Ministry of Justice Frontend makes use of JavaScript.

Having examined these frontend toolkits and reflected upon the lessons learnt at The National Archives in developing the Transfer of Digital Records Frontend, we recommend that Project Omega should adopt the GOV UK Frontend Toolkit, and customise it as needed. An additional advantage of the GOV UK Frontend Toolkit which will aid in the development of Omega from a user testing perspective is its rapid prototype kit[8].

Specifically, we should utilise the HTML, CSS, and minimal client-side JavaScript aspects of the GOV UK Frontend Toolkit, we may also utilise its build system if necessary, however we should not adopt its Server-Side templating.

We have recommended against utilising the server-side templating aspect as it is written in JavaScript utilising a framework called Numchucks which operates atop a Node.Js stack; as previously highlighted earlier in this document, The National Archives has no development expertise in Server-Side JavaScript with Node.JS at this time. Instead we recommend that the server-side templating need instead be fulfilled by Scala and Play in the Editorial Web Application Services layer.

Summary

We have re-evaluated our origin technology choices, and re-recommended technology options that we believe fit well with current and existing investments and skills within The National Archives. We do appreciate that developer recruitment is a difficult process for The National Archives, but we also observe that this is unlikely to be solely resolved by choosing a different technology stack.

Should the The National Archives wish to move to a single stack as an organisation, then we would recommend investigating JavaScript, or better yet, TypeScript, as it could yield two benefits:

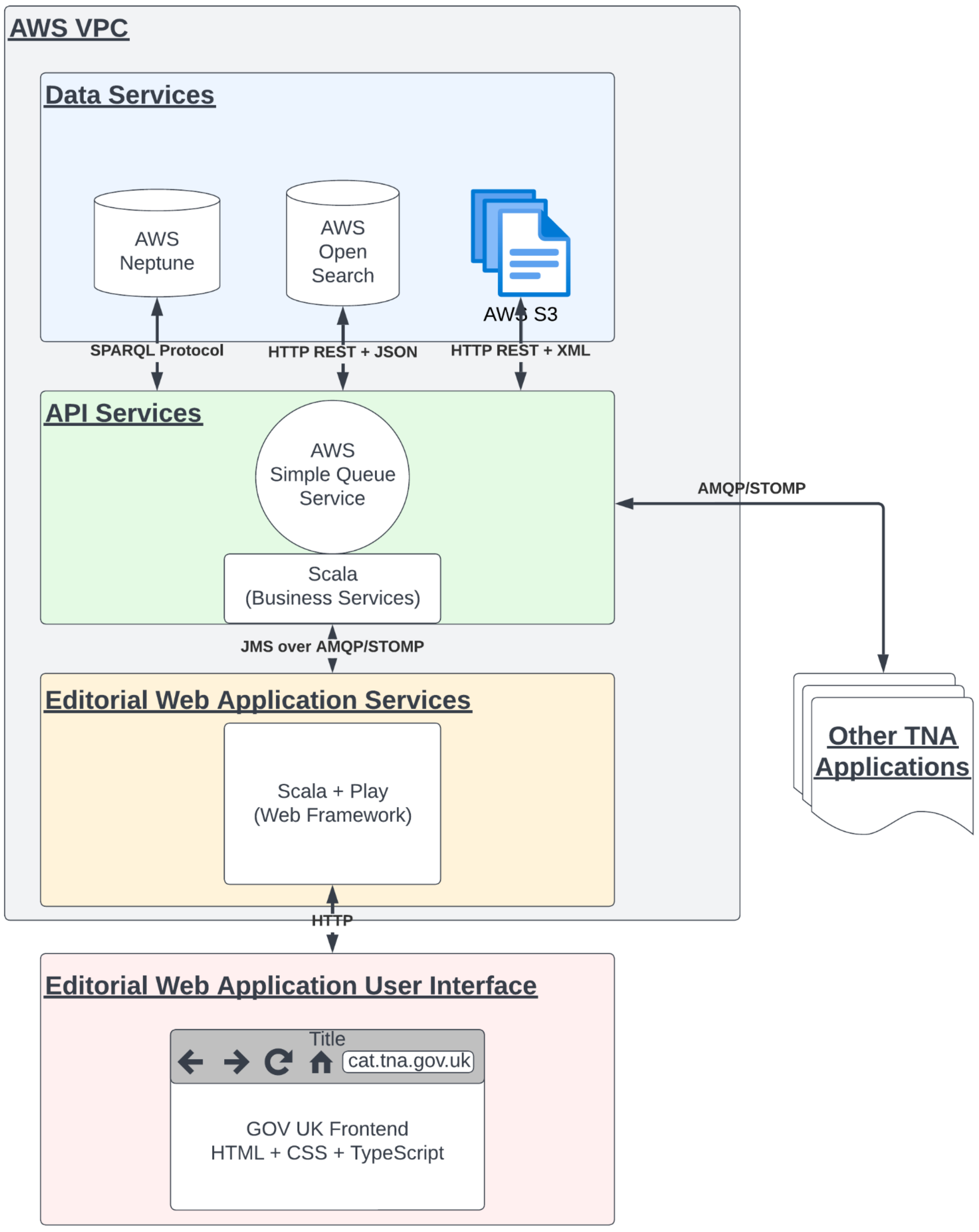
- A single development language for the organisation where all developers across the organisation can more easily collaborate and/or swap teams
- A larger market of talent to draw upon when recruiting

Specifically for Project Omega, our recommendations at this time are:

Role	**Recommendations**
Cloud	Amazon Web Services (AWS)
Provisioning	Terraform, Puppet
RDF Triple Store	AWS Neptune
Full-Text Indexing	AWS Open Search

[8] https://govuk-prototype-kit.herokuapp.com/docs

General File System Storage	AWS Simple Storage System (S3)
Message Broker Fabric	AWS Simple Queue System
Editorial Web Application Services	Play
Editorial Web Application User Interface	GOV UK Frontend
Server-side Programming Language	Scala (or Java where necessary)
Client-side Programming Language	TypeScript (or JavaScript where necessary)


AWS VPC
Data Services
AWS Neptune
AWS Open Search
AWS S3
SPARQL Protocol
HTTP REST + JSON
HTTP REST + XML
API Services
AWS Simple Queue Service
Scala (Business Services)
AMQP/STOMP
JMS over AMQP/STOMP
Editorial Web Application Services
Scala + Play (Web Framework)
Other TNA Applications
HTTP
Editorial Web Application User Interface
Title
cat.tna.gov.uk
GOV UK Frontend HTML + CSS + TypeScript