New Zealand Government

API Standard and Guidelines
Part A - Business

Version 1.0

Approved October 2016
# TABLE OF CONTENTS

- Foreword ........................................................................................................... 8
- Introduction ......................................................................................................... 9
- Scope .................................................................................................................... 10
  - 2.1 Target Audience .......................................................................................... 10
  - 2.2 What is an API? ............................................................................................ 10
  - 2.3 Need/Drivers ................................................................................................ 14
  - 2.4 Why an API Standard? .................................................................................. 14
- Business Context ................................................................................................... 15
  - 3.1 API Evolution ................................................................................................ 15
  - 3.2 Business Change ........................................................................................... 17
  - 3.3 Potential Impacts ........................................................................................... 19
    - 3.3.1 Potential Policy Impacts ........................................................................ 20
    - 3.3.2 Potential Contractual Impacts ............................................................... 20
    - 3.3.3 Potential Organisational Impacts ........................................................... 21
- Principles ................................................................................................................. 21
  - 4.1 Design for the Consumer .............................................................................. 21
  - 4.2 Design Driven Development ......................................................................... 22
  - 4.3 Co Design ........................................................................................................ 22
  - 4.4 Future-Focused Design .................................................................................. 23
  - 4.5 Dogfooding ..................................................................................................... 23
  - 4.6 Stability & Backwards Compatibility ............................................................ 23
  - 4.7 Privacy and Sensitivity Aware ........................................................................ 24
  - 4.8 Well Managed ................................................................................................. 24
- Key Characteristics of an API ................................................................................. 25
  - 5.1 Reference Architecture ................................................................................. 25
    - 5.1.1 Actors ....................................................................................................... 26
    - 5.1.2 Consumption ......................................................................................... 27
    - 5.1.3 Discovery ................................................................................................. 27
    - 5.1.4 Interface Specification ........................................................................... 27
    - 5.1.5 API .......................................................................................................... 27
    - 5.1.6 Application/Logic ................................................................................... 28
    - 5.1.7 Data ......................................................................................................... 28
    - 5.1.8 Context ..................................................................................................... 28
    - 5.1.9 Analytics ................................................................................................. 28
    - 5.1.10 State ....................................................................................................... 28
    - 5.1.11 API Design and Business Process Analysis ........................................... 29

Part A – Business v1.0

Page 3 of 42
5.2 Expose Data as a Set of Resources .................................................................................. 29
5.3 API Publication and Access .......................................................................................... 29
5.4 Discovery ..................................................................................................................... 29
6 Lifecycle Management ....................................................................................................... 31
   6.1 Service Design............................................................................................................. 32
      6.1.1 Consumer Management ...................................................................................... 32
      6.1.2 Service Level Management ................................................................................. 33
      6.1.3 API Design ........................................................................................................... 33
   6.2 Service Transition ....................................................................................................... 33
      6.2.1 Internal Development and Testing ....................................................................... 33
      6.2.2 API Catalogue ...................................................................................................... 34
      6.2.3 API Release Management .................................................................................... 34
      6.2.4 Lifecycle/change management .............................................................................. 34
   6.3 Service Operation ........................................................................................................ 34
      6.3.1 Access Management ........................................................................................... 35
      6.3.2 Consumer Support .............................................................................................. 35
      6.3.3 Incident/Event Management ............................................................................... 36
      6.3.4 API Management .................................................................................................. 36
   6.4 Continual Service Improvement .................................................................................. 36
      6.4.1 Analytics .............................................................................................................. 36
      6.4.2 API Availability ................................................................................................... 36
7 Application of the Standard ............................................................................................... 38
8 Getting Started with APIs .................................................................................................. 38
   8.1 API Components ......................................................................................................... 39
      8.1.1 API Developer Portal ............................................................................................ 39
      8.1.2 API Gateway ......................................................................................................... 40
      8.1.3 API Manager ......................................................................................................... 40
9 API Governance .................................................................................................................. 40
10 Appendix: Glossary ............................................................................................................ 41
TABLE OF FIGURES

Figure 1: Basic API Architecture .................................................................12
Figure 2: Trusted Traveller Example API Architecture ..................................13
Figure 3: API Evolution ..............................................................................16
Figure 4: Current - Integration by Human ..................................................18
Figure 5: Future - Integration via API .........................................................19
Figure 6: API Requirements ......................................................................25
Figure 7: API Reference Architecture ........................................................26
Figure 8: The API Lifecycle ......................................................................31
Figure 9: API Service Lifecycle Management .............................................32
Figure 10: API Core Components & Capabilities ......................................39

TABLE OF TABLES

Table 1 - Common API Categorisation.......................................................11
Table 2 - Trusted Traveller Example API Roles .........................................13
Table 3 - Reference Architecture Actors ....................................................27
Acknowledgements

This document was authored with the assistance of Middleware New Zealand Ltd and the agencies and private companies listed below. Special acknowledgement must be given to the Ministry of Business, Innovation and Employment upon whose work the technical content of this standard largely stands.

Authors

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swithin Foote</td>
<td>Integration Architect</td>
</tr>
<tr>
<td>Tina Groark</td>
<td>Business Architect</td>
</tr>
<tr>
<td>Josie Shaw</td>
<td>Solution Architect</td>
</tr>
<tr>
<td>Chris Solomon</td>
<td>Identity Architect</td>
</tr>
</tbody>
</table>

Contributors

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neville Atkinson</td>
<td>Solution Architect, NZQA</td>
</tr>
<tr>
<td>Dan Cooper</td>
<td>Lead Enterprise Architect, MSD</td>
</tr>
<tr>
<td>James Fitzsimons</td>
<td>Lead Architect, Market Services, MBIE</td>
</tr>
<tr>
<td>Andrew Kirkland</td>
<td>Senior Integration Architect, DIA</td>
</tr>
<tr>
<td>Grayson Mitchell</td>
<td>Solution Architect, MoE</td>
</tr>
<tr>
<td>Andy Neale</td>
<td>DIA</td>
</tr>
<tr>
<td>Barry Polley</td>
<td>Principal Advisor Business Technology, MPI</td>
</tr>
<tr>
<td>Oliver Seiler</td>
<td>Solution Architect, MBIE</td>
</tr>
</tbody>
</table>

Consulted

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simon Ferguson</td>
<td>Principal Advisor Services Transformation, MBIE</td>
</tr>
<tr>
<td>John Gonzales</td>
<td>CEO &amp; Principal Consultant, Hashbang</td>
</tr>
<tr>
<td>David Hay</td>
<td>Product Strategist, Orion</td>
</tr>
<tr>
<td>Alastair Kenworthy</td>
<td>Principal Sector Architect, MoH</td>
</tr>
<tr>
<td>Andrew Kirkland</td>
<td>Senior Integration Architect, DIA</td>
</tr>
<tr>
<td>Brian More</td>
<td>Lead Principal Enterprise Architect, Westpac</td>
</tr>
<tr>
<td>Lein Ton</td>
<td>Enterprise and Solution Architect, Plunket Plus</td>
</tr>
</tbody>
</table>
Foreword

The Government ICT Strategy and Action Plan is founded on principles of information sharing and collaborative service delivery across the public, private and NGO sectors.

Government customers – individuals and businesses – will benefit if they can receive services and fulfil obligations with a minimum of effort through the most convenient channel. Ideally, customer’s transactions will be invisible by-products of their day-to-day interactions with public and private sector services. This approach can also reduce unnecessary investment by agencies in dedicated service channels and rich user interfaces that would be better placed elsewhere.

The strategy further calls for government information to be open wherever possible and available as a resource for research to enhance the lives of all New Zealanders.

To achieve these outcomes, government agencies need to expose their information and services to each other and to authorised third parties, while honouring their obligations to protect individual privacy and commercial confidence, and to give customers control over the use of their information.

Web-based application programming interfaces (APIs) are a cornerstone of this strategy, as the preferred mechanism to offer direct access to government information and transactional services.

These API Standards and Guidelines were commissioned and endorsed jointly by the Government Enterprise Architecture Group and the ICT Partnership Framework Technology Working Group. Their primary intent is not to constrain the use of APIs in government, but rather to give agencies and vendors some common, default guidance on API implementation and thereby accelerate the provision of government APIs.

This document will not stand alone, but complement other work including the collaborative service architecture, identity management reference architecture, data standards and other artefacts currently in development.

Version 1.0 of this document is now published and available for agency use. Over the next six months we will be seeking commentary on its usability and effectiveness, with an expectation of a revised version after that. Feedback, please, to GEA@dia.govt.nz.

I would like to thank the many agency representatives who helped us and Middleware NZ to produce these guidelines, in particular the Ministry of Business, Innovation and Employment, whose API guidelines and strategy formed its foundation.
# Introduction

In 2012, the Government announced ten priorities for government - the Better Public Services (BPS) programme. One of the five BPS Result Areas is about improving interaction with government – by making it easier and more efficient for customers to deal with government, and by providing government services in new and different ways.

Both individual and business customers have told us they find government services complex and fragmented, and that dealing with government takes far more time and effort than it should. For business customers, especially small businesses, this has a direct cost - time spent on administration is time away from growing their business. For individual customers, the cost is less obvious – they struggle to access the services they need and often aren’t aware of the services available to them.

Additionally, digitally savvy customers are demanding and expecting more convenience, more information and more personalisation than ever before. As customers get used to intuitive, seamless and secure user experiences from firms like Amazon, Uber, and Apple, government is being challenged to deliver customer experiences to meet these increasingly sophisticated expectations.

To respond to this combination of factors, government is moving towards a customer-centric service delivery model, where the needs of the customer are considered alongside (or ahead of) the needs of any one agency. Customers may need to interact with multiple agencies in order to achieve their goal (e.g. setting up a business), and customer-centric service delivery aims to view public services as a joined up experience, which in combination can achieve customer goals. This increased focus on customer needs requires agencies to "open up" their services so they can be combined in new and creative ways in response to emerging customer demands.

Additionally, government is also moving towards being a wholesaler of government services, rather than just a retailer. That is, Government is actively encouraging 3rd parties to use government services to create value added services for their own customers. "Opening up" agency services to 3rd parties will encourage commercial organisations and NGOs to act as intermediaries for government services, and has the potential to dramatically improve interactions with government. For example, MBIE has "opened up" the New Zealand Business Number (NZBN) register. Some organisations are using this service to directly maintain the primary business data (PBD), which saves them having a separate manual interaction with MBIE via the web site every time it changes.

Today most agency services are not “opened up” and have limited integration capabilities. Agencies need to learn how to provide their services in new and different ways. This document focuses on one of these new ways, specifically, using Application Programming Interfaces (APIs).

APIs can be viewed as simply another channel for delivering services, like websites, call centres and shop-fronts. APIs underpin many of today’s websites and mobile apps, and are becoming increasingly important in the digital economy. This is because APIs can be easily incorporated into other software to deliver new and innovative services for customers and for the public sector itself. APIs can aid system builders and integrators in building a more joined up government.

Agencies need focused guidance, approaches and techniques to help increase their knowledge and skills in this area. There is a wide range of knowledge and skill regarding API design and delivery across government agencies, from small niche organisations making their first attempt at information sharing to large agencies who have been delivering APIs in one form or another for the last few years (e.g. MBIE have been providing APIs since 2002). Hence this document tries to offer a balance of guidance for those new to APIs along with quick lookup standards, which should assist agencies in achieving consistency and commonality in their API deliverables.
2 Scope

The document aims to provide a set of high-level standards with design and implementation guidance, along with low-level API best practices to guide government agencies in their development of APIs.

Part A looks at the business context for APIs within government and articulates the principles and considerations that could impact an agency when creating APIs. It looks at APIs in the context of their impact on the organisation as well as across government and public services through to commercial innovation.

Part B contains the technical implementation details, including patterns for API security and general API implementation standards for API developers & consuming application developers.

While primarily oriented towards REST APIs, Part A contains guidance that is applicable to traditional SOAP services as well. However, the definitive technical standard for Government for SOAP services remains the GEA-NZ Secure Web Services Standard.

The document will use hypothetical or actual use cases with a government context to illustrate practical application of the concepts described.

2.1 Target Audience

The target audience for this document is primarily enterprise architects, solution architects and API developers within, or contracted to, agencies. It may also be of interest to commercial entities, NGOs or other third parties who are developing, or planning to develop, applications which use government APIs.

2.2 What is an API?

An API is a piece of software, which provides a way for other disparate pieces of software (applications, systems) to talk to one another. It is:

- A means of requesting and retrieving information from a system/server. so that it can then be presented to the user in a context/process-specific way, e.g. in a mobile app or web application
- A way of de-coupling system to system interactions
- Not tied to any specific programming language
- A contract between the application developer and the provider describing what information or functionality they can gain access to and how that access is provided

APIs are commonly categorised as shown in the table below:
<table>
<thead>
<tr>
<th>API Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal API</td>
<td>An API that is used solely within your organisation by known internal personnel (employees or contractors). For example, an API that provides revenue forecasting information.</td>
</tr>
<tr>
<td>External API</td>
<td>An API that is used by customers and/or application developers who are outside your organisation.</td>
</tr>
<tr>
<td>Partner API</td>
<td>An API that is used by business partners who are known to your organisation and with whom you often have a special contractual relationship. For example, an API that provides sales information to the partner.</td>
</tr>
<tr>
<td>Public or Open API</td>
<td>An API that is used by anyone who wants to use it, as long as they can meet your access control requirements.</td>
</tr>
</tbody>
</table>

*Table 1 - Common API Categorisation*

Below is an illustration of the basic architecture of an API and its usage:
From right to left: There is an **API Provider**, who owns the resource (information or functionality) in **Applications** to be exposed by the API, and whose Developers build the API (**API Developers**). API Developers start by defining the **Interface Specification**, then they build the API. Next, there is an **Application Developer** who creates new software, or adapts existing software, to use the API. Once complete, this software is known as an **API Consuming Application**, which uses the API to gain access to the resources the API Provider is offering. Finally, **Customers** use the API Consuming Applications. The Application Developers, the API Consuming Applications and the Customers are referred to as **API Consumers**.

For example, in order to enable business travellers to transit the border between Australia and NZ quicker, a trusted traveller system needs to be designed. This will enable businesses with frequent flyers that know and understand the Customs and biosecurity requirements to experience a faster arrivals process on entry into NZ.
Figure 2: Trusted Traveller Example API Architecture

<table>
<thead>
<tr>
<th>API Role/Component</th>
<th>Trusted Traveller Example Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Provider</td>
<td>MPI develops a Trusted Traveller application that collaborates with government agencies and private sector organisations to determine whether a passenger is eligible for Trusted Traveller status or not.</td>
</tr>
<tr>
<td>Application</td>
<td>The Trusted Traveller application.</td>
</tr>
<tr>
<td>API Developer</td>
<td>MPI develops an API to allow devices, such as smart gates at airports, to validate the eligibility of passengers for Trusted Traveller express transit.</td>
</tr>
<tr>
<td>Application Developer</td>
<td>Smart gate developers enhance their devices to call the API and develop business rules for Trusted Travellers.</td>
</tr>
<tr>
<td>Consuming Application</td>
<td>Smart gates at the airport use the API to determine whether the passenger is eligible to be released through Trusted Traveller express transit.</td>
</tr>
<tr>
<td>Customer</td>
<td>Passengers use smart gate at the airport to cross the border and gain Trusted Traveller express transit.</td>
</tr>
<tr>
<td>Consumers</td>
<td>Smart gate developers, smart gates and passengers using those smart gates.</td>
</tr>
</tbody>
</table>

Table 2 - Trusted Traveller Example API Roles

Clearly, some APIs will be for internal government use (e.g. information exchange) whilst others will be for external/public use. The types of information offered by these different types of API will be limited by the privacy act and security concerns.
2.3 Need/Drivers

So why build an API? In what situation is an API the right thing to develop? There is an increasing drive towards government agencies delivering APIs as part of open data and ease of digital interaction with government. But different government organisations have different drivers:

- **Moving towards a collaborative model of service delivery** – enabling agencies to participate in external business processes;
- **Improving customer interactions with government** – enables integration with partner agencies at a system level rather than via humans;
- **Moving from retailer to wholesaler of services** – supporting integration with 3rd parties and enabling them (rather than the agency) to build the customer-facing aspects;
- **Using Cloud services** – ensuring reliable and secure information exchange with Cloud-based COTS solutions;
- **Improving access to information** – providing a more robust interaction model than the current practice of data/screen scraping;
- **Meeting the goals of Open Data** – using APIs to open public data to public access, enabling private sector to create their own solutions based on public data;
- **Improving quality of information** – providing a reliable single source of the truth and a more robust interaction model than the current practice of regular bulk information sharing;
- **Exploiting emerging technology** – encouraging uptake of new and innovative technologies to keep pace with customer demands and expectations;
- **Enabling channel shift** – supporting agencies in their desire to make APIs their main web channel in preference to developing agency-specific web user interfaces;
- **Contributing to the Business Growth Agenda** – helping to grow competitive businesses within New Zealand in the information-based economy.

2.4 Why an API Standard?

Why is an API standard necessary for government organisations?

Whilst some NZ public sector organisations have been producing APIs for a while, others are taking their first steps in offering APIs for public use. It is useful to provide a reference and set of guidelines to support this API development in order to:

- Encourage consistency across government API offerings and thus making it easier for third parties to build solutions using government APIs
- Avoid different government organisations developing or delivering idiosyncratic APIs
- Generate awareness of what is needed to support APIs as a delivered product to private sector customers e.g. levels of service, support capabilities etc.
- Capture API best practice for reference by the API development community, indicating where commonality is required and which areas allow for more flexibility
- Reduce disruption to application developers as government APIs evolve
- Reduce effort for agencies by simplifying the process of developing and delivering on their API strategies
3 Business Context

The government ICT strategy paints a picture of agencies partnering with private sector in order to unlock the value of information to help solve complex problems and generate innovative ideas. The development of APIs is considered a key part of this picture in order to improve access to public services and deliver a joined-up experience to the customer. This approach requires collaborative service delivery, where front end and back end capabilities may be developed separately, part by the government sector, part by commercial entities or NGOs, and delivered in a coordinated fashion to offer a seamless experience to the customer. This change in approach can obviously impact the business function of an agency.

One recent NZ Government example of collaborative service delivery is the new IRD service that allows customers to file their GST returns directly through some software packages. IRD worked closely with software package providers to design this service, and software package providers extended their packages to use this service. Now, for customers using this capability, their GST return information is prepopulated from the sale and purchase figures from the software package. The customer no longer needs to manually enter their figures into their GST return or print a copy and send it to IRD. It all happens in a seamless interaction from the customer’s perspective, reducing the amount of time they spend double handling their accounts information and reducing opportunities for error.

Businesses who trialled this service say it was convenient, easy to use and reduced the amount of time they spent “double handling” accounts information.1

If we look at API delivery in an all of government context we see that the initial focus tends to be within an agency’s sphere of influence, as part of the business systems they deliver and processes they perform. APIs are seen as a means to an end, quite often driven from a technological perspective. Whilst organisations across government have different levels of take-up in terms of APIs, there is a common pattern of evolution, which can help when developing an API strategy and road mapping delivery of APIs.

3.1 API Evolution

An organisation’s development of APIs tends to start from an internally focussed perspective, concentrated on existing internal resources (data, functionality). But over time this perspective broadens to encompass the needs of their sector, then all of government and beyond. A potential roadmap for the evolution of government APIs over time is illustrated below, aligned to examples from mobile banking’s evolution:

Initially, parochial APIs are developed to enable programmatic access to existing information stores (e.g. see radio spectrum usage) or to request an action from an agency, which spawns an internal business process (e.g. request patent renewal). These are built from the perspective of the resources the agency holds, not from an all-of-government perspective. High transaction volumes are often a driver for agencies developing parochial APIs. The APIs are designed either for consumption by systems within the organisation or by selected consuming applications outside the agency. Access to information may also support some filtering (e.g. view available/free radio frequencies). This step of the evolution may be bypassed by organisations whose drivers are focused on external integration.

Opening up APIs for integration is the next step, where business processes which span government agencies are acknowledged and supported. This often encompasses a move from batch processing to real time data update, with the associated mind-set and business process change. APIs form another digital channel, alongside the standard web UI channel that supports the majority of interactions, with the paper channel being retained only to support exceptions.

APIs are then created which start to support authorised modification or addition of information to information stores. This is either to support direct update by the end customer (via web form), or by third parties acting on their behalf (with delegated authority, e.g. mobile apps). APIs are at this point becoming one of the core products that an agency releases, offering transaction handling and taking over from web interfaces. Ideally private sector organisations increasingly use the APIs to incorporate public sector capabilities into their commercial offerings, migrating government from the role of retailer to that of wholesaler.

Next, APIs become the primary channel for information and request for action. This fosters innovation by opening up resources to third parties who have not previously had access, such as commercial entities & NGOs. This stage of API evolution starts unlocking the value of information and enabling compliance or interaction with government processes to be built invisibly into everyday applications (e.g. automatic GST returns via Accounting software). It also moves the onus of UI development from government to the marketplace, allowing new and intuitive means of access from mobile devices, smart phones, domestic appliances etc. to be created.

Finally, we move to the automatic use of APIs, where APIs are both machine discoverable and consumable, requiring minimal developer input for effective delivery of
capability to the customer. It is difficult to predict the impact this will have, as there are no examples to date of this kind of capability.

3.2 Business Change

The building of an API channel opens up an agency to business process change which influences the ways they work internally and with partner organisations. It can also alter the main point of contact with the public, or add a layer of abstraction, as API consuming applications replace direct contact with customers. Hence, change management will be paramount in the development of this new API channel.

For instance, MPI are looking to develop an API to assist animal status declaration, which keeps track of cattle and sheep transportation. Currently animal transportation information is captured in paper form by the farmer/transporter, which must be produced on demand when MPI needs the information for a biosecurity response. However, if the information can be captured via mobile application and passed to MPI through integration with the farm management system, then the information is more trustworthy and will be readily available to support associated MPI business processes. Physical spot checks could become a thing of the past, so MPI no longer have direct contact with the customer (farmer/transporter), but only the API consuming applications (farm management system). This has an impact on internal team make-up, working practices, travel and interactions. It is important when designing APIs to consider the impact on the business and to put in place the change support that will inevitably be needed to underpin controlled business process change.

Whilst most government organisations are looking at API development from the perspective of their own needs or the needs of their sector, there is an emerging consideration of APIs in the context of all of government integration (federated service delivery) and joined-up capability to support customer-centric public services. Once the fundamental building block APIs are in place to assist with integration and information sharing, it enables commercial operators to innovate and combine multiple services to provide value-add customer-focused end products. Is it possible to foresee a future where public interactions with disparate government agencies are automatic, seamless, and built around the end-user’s ultimate goal (e.g. setting up a business)?

Currently a member of the public acts as the integration point between agencies, supplying the same or similar information to each agency (name, contact details) whilst trying to achieve their objective and having to develop a mental model of each agency’s business practices to navigate by.
With the gradual emergence of an API layer across government, the need for this human information sharing mechanism should reduce, moving the integration into the system layer. Coordination across this layer will be fundamental to seamless delivery of federated services.²

² See DIA’s work on FSD for more details
There is also a move towards government as the wholesaler rather than the retailer i.e. provider of the information at the programmatic layer (API), rather than the presentation layer (e.g. web site). If the APIs were available, software packages that customers already use could be enhanced to complete many of the obligatory government interactions automatically, in the background (with customer permission). The commercial world is seen as having the means, experience and drivers to develop effective ways of interacting with the public, so by opening up resources via APIs government agencies can encourage innovation and create opportunities for more intuitive and supportive channels of working for the customer. For example, MBIE currently provides APIs which allow 3rd parties to manage their customer’s NZBN primary business data and are working on a business registration service that allows business customers to complete their common government registrations in one place.

There will still be instances where government needs to act as an impartial intermediary and offer capabilities which are unbiased and independent (e.g. patent examination), or offer basic capabilities for organisations which cannot afford anything complex, but these should be the exception rather than the norm. It is also recommended that any action which legally commits the customer (e.g. GST return submission) require a manual step so that such commitments are customer controlled.

### 3.3 Potential Impacts

The building of an API channel has the potential to impact an agency in a variety of ways. Some of the areas potentially impacted are discussed below. The list presented below is by no means complete.

---

3.3.1 Potential Policy Impacts

- **Legislation / regulation** – APIs must align to the legislative or regulatory requirements on the agency. In some cases, legislation and/or regulations may limit the opportunities for agencies to use APIs. For example, legislation may not allow for electronic submission. Conversely, an API may introduce new risks that are best managed through legislation or regulation. For example, if there is a bug in a software package which means the information submitted electronically is incorrect, who is liable for the error and the resulting penalties or criminal offence, if any?

- **Privacy** – Another area of legislation that APIs must align with is the Privacy Act. Agencies with APIs that involve personal information will need to consider the privacy implications. If appropriate, an agency may choose to undertake a privacy impact assessment to understand the privacy implications of an API. For example, the API may introduce the possibility for amalgamating information in ways unpredicted by the agency. Additionally, the Privacy Act may impose limitations on APIs. For example, the methods available for uniquely identifying individual customers are limited by the Privacy Act.

- **Identification** – Does the agency need to (uniquely) identify any, or all, of the API consumers? What is the appropriate mechanism for uniquely identifying the customer who is registered via an API consuming application?

- **Risk appetite** – APIs have the potential to introduce new risks to an agency and/or remove some risks. Different API designs may have different risk profiles, which need to be understood.

- **Business analytics** – One side effect of APIs is that more and more information is exchanged electronically, potentially creating new information stores. These new information stores provide new opportunities for amalgamating personal information and sharing information.

- **Terms of use** – What terms of use are appropriate for your API? Consideration should be given to the purpose for the information exchanged via the API and the limitations on its use by the application developer for other purposes. Additionally, consideration should be given to how compliance to the terms of use will be enforced and monitored.

- **Pay-per-use** - Whilst it is possible to consider pay per use charging for APIs where commercial entities will be making money from the services offered, pay per use can have a negative impact by impeding take-up. Even requiring a one-off charge as an on boarding fee can impact take-up especially by smaller application developers and NGOs. There may also be some legal restrictions to charging for use of APIs. (Pay per use should be differentiated from transaction fees or official fees.)

3.3.2 Potential Contractual Impacts

- **Creating new contracts and MOUs** – Commercial use of APIs may require more rapid contract and Memorandum of Understanding (MOU) development to encourage uptake and to help ensure all the intended consumers are supported. Contractual arrangements that take too long to organise or have onerous conditions will discourage commercial uptake.

- **Extending existing contracts and MOUs** – A new API may have implications for existing contracts or MOUs. As the integration approach has the potential to change, so too does the contract or MOU.

- **Structure of contracts and MOUs** – The structure of a contract or MOU may need to change in line with the API development approach. For example, if the development approach is agile, with frequent API releases, then the structure of the contract or MOU should have the flexibility to easily allow changes to maintain alignment with a rapidly evolving API.

- **Enforcement of MOUs/contracts** – APIs often make information available to consumers in the context of specific terms of use. For example, personal information about the customer is made available from the API to consuming applications for a specific purpose. What are the implications if the application developer then uses the information for other purposes?

- **Service levels** – APIs often require service level agreements (SLAs) with application developers and suppliers. SLAs are important to understand and will, likely, evolve over time as the API matures. MOUs and contracts should cater for this scenario.

- **Coverage of MOU** - Existing MOUs for information sharing may be sufficient, initially, to cover an application developer’s planned use of an API, but once an API is exposed new developers not covered by the MOU may request access. In this situation, careful consideration must be given to whether the MOU should be modified or access to the API denied.
• **Extent of Sign-up** – When an API is offered publicly, and application developers register for use, it is important contractually to know whether the individual is signing up to an API in their own capacity or on behalf of an organisation.

• **Onward information handling** – It may be appropriate to contractually impose limitations on the on-going use of information exposed via APIs. Application developers can cut, amalgamate, derive metrics and present information as they choose, once they have obtained data via an API unless you impose limitations.

• **Responsibilities definition** – Agencies may wish to clearly contract responsibilities with application developers to ensure customers of their applications are made aware of their legal obligations in regard to use of agency APIs (e.g. changing tax information via their application).

• **Information currency** - Some application developers may be tempted to use the API to update their own local caches of information. Depending on the information update period, this can be risky as it means they are working from potentially obsolete information.

3.3.3 **Potential Organisational Impacts**

• **New services** – if this is an agency’s first foray into APIs there may be a completely new model for service delivery, potentially with the need to support new types of customers (e.g. software developers) and product release cycles. It is important to consider the organisational implications of this new model.

• **New processes / procedures** – In addition, APIs may introduce new ways of working, the implications of which need to be understood. For example, if the API is automating a manual process then the implications of this new automated process need to be considered.

• **New skills / techniques** – An API channel for service delivery requires a new set of skills and techniques to manage the application developers and the communications with application developers e.g. incident resolution.

• **Service delivery partners** – An API channel opens the opportunity to develop relationships with 3rd parties who deliver services on behalf of the agency. These service delivery partners may need to be engaged and managed in new and different ways.

• **Availability** - The availability of an API should be no less than the availability of the equivalent agency online service or website, as this will have already set an expectation from the user community. An API may need to be available 24/7, but consideration should be given as to whether support for the API also needs to be 24/7.

• **Service levels** – Opening an API channel may necessitate an agency to offer a level of service, where previously there has been no such need. Commercial experience and knowledge could be required in order to define realistic and achievable service levels that serve everyone. These service level changes may require additional staff to cover increased support hours, for example. There is also a need to measure, monitor and report back on performance to ensure SLAs are met.

• **Support services** – There may be a reduced need for agency support services, where consuming applications replace agency presentation layers, moving customer calls from agency support staff to the application developers.

• **Principles** - In order to incorporate All of Government (AoG) thinking into API development it is useful to have a set of business principles which should influence the way APIs are built and their usefulness over time.

4 **Principles**

4.1 **Design for the Consumer**

In alignment with the drive of the BPS Result 10 to become more customer-centric, APIs need to be developed with the customer primarily in mind. The application developers also need to be considered because, of course, if the API doesn’t meet their needs then they won’t develop consuming applications that use it. This leads to APIs being developed as generic as possible in order to meet the basic needs of all potential consumers. API developers should not try to predict how the customers will interact
with consuming applications, but should leave the application developer to innovate. By designing APIs for the consumer, agencies are likely to build APIs which are intuitive and easy to use, thus ensuring uptake of their APIs and encouraging access to public information.

Initially, application developers will be the main users of APIs. Hence development and delivery of APIs should be geared around making it as easy as possible for developers to discover, understand and develop against those APIs. So APIs, along with the associated on boarding and support processes, should be simple to understand and well described.

Some examples of this principle in action:

- Ensure a low barrier to entry so it is easy to start using the API.
- Provide sandbox APIs so application developers can try out APIs and develop in parallel.
- Be responsive to feedback and bug reports.
- Provide automated on boarding processes, as manual processes can limit take-up.
- Provide prototyping tools and support for development.
- Ensure funding cycle does not introduce a bottleneck for uptake within government.
- Sell the concept to the customer early on so they create demand which industry need to meet.
- Create an atmosphere of competition in industry for take-up of the API.
- Create an SDK to support an agency’s APIs, including examples.

### 4.2 Design Driven Development

This builds on the principle of "Design for the Consumer" and could also be phrased as "Specification Driven Development" because the interface specification for the API is written first. Designing the interface does not require full design of the code which will sit behind it. The interface can be in place without the functional capabilities being built yet still be useful. By defining the interface first, there is a known integration point, which both the application developer and API provider can refer to and build to. Potential application developers can use the specification to assess whether the API will meet their needs, or if further information or parameters are essential to them. The process of creating a specification before development forces the API developer to properly consider the purpose and content of the interface they are providing, before coding starts. Taking a design driven approach simply means that the API design should be performed before development begins; it does not mean that development is hindered in any way. There are a number of tools that assist with this process, however the tool to create the specification should be of little significance, the act of creating the specification is what counts.

### 4.3 Co Design

This principle is the natural extension to both "Design for the Consumer" and "Design Driven Development" principles and includes collaborating with the consuming application developer community when designing the interface. This is important to help meet Result 9 aims of making business engagement with government more efficient, not just for business as a customer of government but also where business works together with government. Co Design works as an enabler to business innovation and supports better interaction with government.

Application developers are one of the primary customers for the API and, as such, are a primary source of requirements. Working with application developers to define the requirements and to design the interface ensures the API will meet their needs. However, identifying potential application developers and understanding the demand from the market can be difficult. Careful consideration should be given as to the best approach to engaging with the market and identifying your potential application developer community.
Additionally, it is strongly encouraged that application developers start to produce API consuming applications based on the interface specification as early as possible. This agile or iterative approach helps ensure real-world feedback is incorporated into the API design as early as possible. Completeness is not necessarily the goal, especially in initial APIs. The goal should be to get early partially complete releases out, defining the limited capability they offer, to enable consuming application uptake. Development needs to be flexible and agile to adapt to early adopters’ feedback in identification of pitfalls and issues. However early releases should be tested and stable so as not to impede uptake. The aim is to try, then adapt, rather than waiting to release a fully functional API.

4.4 Future-Focused Design

Most agencies will have a variety of legacy systems that they need to continue to support and service. It is important to remember that business, technology and application architects should be designing for the future of their organisation, and not "hamstring" their APIs under development by designing them to work in the way the legacy system currently works, or to tailor APIs so that they work perfectly with all legacy systems. The aim is to be future-focused (whilst still pragmatic) and develop APIs to meet future needs.

These needs will be agency-specific, but may include a drive to move to a wholesaler rather than a retailer role, where the agency reduces its web presence in favour of enabling commercial entities to create applications, or add richness to existing applications, through access to agency APIs.

4.5 Dogfooding

When building an API, there is always a danger of building the wrong thing in the wrong way for the wrong people. This is especially a risk in the absence of external developers driving or informing the development of the API. The simplest way to ensure an API is useful and consumable is to build the API as part of delivering a larger-scale internal business support system such as a web application. In this way, the customer and application developer are internal, and the first principle "Design for the Consumer" applies.

The best way to design an API is in collaboration with the potential consuming application developers. The best way to understand and address the weaknesses in an APIs design and implementation is to use it in a production system. So, wherever feasible, design an API in consultation with application developers but build it in parallel with an accompanying integration of that API to ensure it is fit for purpose, i.e. build for inside use, then expose. This has the added advantage of meaning that the cost of API development is covered throughout the build of the business support system or integration mechanism, reducing the need for extra funding.

4.6 Stability & Backwards Compatibility

It is important that APIs have stability (are available and work consistently) and support a velocity of change which is acceptable to the application developers. Early versions of APIs should be available via pilots or on developer portals so application developers can work them and identify areas of instability before an API goes into production.

Application developers will not always be able to adapt to new capabilities or changes to existing interfaces as quickly as the API providers might wish, due to organisation priorities and funding. Hence minor changes to APIs must always be deployed as fully backwards-compatible upgrades. For major changes, which are not backwards compatible, the old API version should be maintained alongside the new version for an appropriate period to allow all-consuming applications to transition. By monitoring usage, it should be possible to assess when an API version can be deprecated, either because it is no longer being used or because the usage pattern does not warrant maintaining that particular version. In any case, it is important to clearly communicate
with your developer community and manage expectations as to longevity of a particular version of an API.

4.7 Privacy and Sensitivity Aware

APIs are used extensively for passing information, so it is important to consider the information privacy and sensitivity aspects of data being passed to and from APIs to ensure that data is protected adequately. Consideration should be given as to whether a privacy impact assessment\(^4\) and/or a security risk assessment\(^5\) is appropriate for the API during each stage of development, from concept through to implementation. For example, if the API is providing programmatic access to publicly available information, the privacy considerations are likely to be minimal and the security considerations limited to the usual suspects, like the OWASP top 10. However, privacy and security considerations become hugely important if the API is providing programmatic access to private personal information. In this situation, it may be appropriate to do regular assessments, especially early in the concept phase to ensure any privacy or security constraints are understood before design.

There is also the issue that with ease of data consumption comes increased ability to combine data from different sources, which increases privacy risks and the potential for unintended information leakage.

4.8 Well Managed

As API take-up increases over time, there is more likelihood of API change, redevelopment and adaptation to meet new needs. A well-managed API should handle regular change. API management gives an agency the capability to deliver good version control capability and forewarn application developers of changes which may impact them, including failures and outages. For this reason, it is important that all application developers are identified, even if the API is considered public.


\(^5\) [https://www.ict.govt.nz/guidance-and-resources/information-management/privacy-and-security/]
5 Key Characteristics of an API

At its heart an API is, as its acronym suggests, an application programming interface. This simply means that an API is an interface that enables software inter-communication. The purpose of these interfaces is to enable safe and reliable communication in a standards-based way.

An API should strive to deliver the following functional and non-functional requirements:

- Security
- Usability
- Reliability
- Testability
- Scalability

![Figure 6: API Requirements](image)

5.1 Reference Architecture

The content in this document is based on the following reference architecture. The diagram below is an illustration of the architecture and the concepts are further expanded, layer by layer.
Figure 7: API Reference Architecture

Trust exists between individual layers, but decreases the further up the model you go, with a high level of trust in the lower layers but minimal trust in the upper layers.

5.1.1 Actors

<table>
<thead>
<tr>
<th>Actor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>This is the person who uses a consuming application.</td>
</tr>
<tr>
<td>Consuming Application</td>
<td>This is any application or device that consumes an API (interface) that you provide. They can be defined as anything that uses your API.</td>
</tr>
<tr>
<td>Application Developer</td>
<td>An application developer in an API context is a developer that creates applications that consume your APIs. An application developer needs to be able to discover, understand and access your APIs. Application developers can be internal to your agency, developers that work with trusted partners, developers from other agencies or developers from community.</td>
</tr>
<tr>
<td>API Developer</td>
<td>An API developer creates and documents an agencies APIs. API developers generally have a good understanding of an agencies</td>
</tr>
</tbody>
</table>
### Table 3 - Reference Architecture Actors

<table>
<thead>
<tr>
<th>Actor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Owners</td>
<td>The product ownership function usually resides in a business area rather than technology. The role of the product owner is to understand the product that the agency is trying to deliver and be able to make decisions on the representation of the product in an API.</td>
</tr>
<tr>
<td>API Product Manager</td>
<td>The product manager is usually a technical role. They understand an agencies API landscape and are owners of API management platforms.</td>
</tr>
</tbody>
</table>

#### 5.1.2 Consumption

Relates to the Usability functional requirement.

Consumption refers to use of the interfaces that are exposed. Consuming applications can be anything that uses an API. Consuming applications “drive” your APIs.

Application developers build the applications that use your APIs. When designing an API, the usability or function of the API should be aimed at this group.

This component has a low level of trust.

#### 5.1.3 Discovery

Relates to the Usability functional requirement.

Interface specification must be published for it to be discoverable. API discovery refers to the way that your APIs are understood by an application developer. Application developers like to see an API or interface specification in a format that they are familiar with. The functional and non-functional “behaviour” of the API should both be well described. Refer to the API Development section in Part B for details.

#### 5.1.4 Interface Specification

Relates to both the Usability and Security functional requirements.

An API interface specification is a technical description and reference model for an API. The specification is used most commonly by application developers to understand the technical aspects of an API. However, interface specifications should also be provided in a machine readable format. Refer to the API Development section in Part B for details.

#### 5.1.5 API

Relates to all functional and non-functional requirements.

The API component of the reference architecture is multi-functional. It provides API authentication and authorisation, preferably using a standard approach. For more details on API security see the API Security section in Part B.

To protect an agency’s resources, the API component should perform threat protection and quality of service (QoS) functions. This could be in the form of “throttling” requests, schema validation and common vector attack protection (SQL injection, Cross Site Scripting etc.). This functionality should not be seen as a replacement for other defences, but an augmentation to those defences.

In some cases, simple orchestration may be performed within this component. An example of simple orchestration is a “mashup” of two back end services or applications into a single API.
The API component should be responsible for maintaining interface specifications and publishing these to the Interface Specification component.

In a bimodal IT model this component emphasises agility and speed.

This component has a medium level of trust.

5.1.6 Application/Logic

This component is where complex business logic is handled. In many cases this will be a system or group of systems that exist at an agency that do not conform to a common standard. This component is where fine-grained authorisation should be performed based on business logic, which that the API component should have no knowledge of. In a bimodal IT model this component emphasises safety and accuracy.

This component should verify the authenticity of the API component, referred to as "platform authentication".

This component has a high level of trust.

5.1.7 Data

The data component represents one or more sources of data within an agency. It does not necessarily need to be a traditional data store or database; it simply represents the fact that an agency will inevitably have data that can be represented in an API.

This component traditionally verifies the authenticity of the application component, referred to as "platform authentication".

The data component has the highest level of trust.

5.1.8 Context

Context can take many forms and it is simply a way of providing contextual information for the request to downstream components that may require it. The main purpose of context is to prevent layers of the architecture having to lookup information that has already been gathered elsewhere by something that the component trusts. A good example of this is where the API component has performed authentication and gathered role information for the customer. If the application components need to perform fine-grained authorisation, the API component can pass customer role information as context for the request.

5.1.9 Analytics

Analytics are key to the success of an API program. Technical analytics enable you to maintain API stability and performance while business analytics enable you to grow your API program. Performance analytics enable you to ensure your APIs are meeting SLAs and other contractual obligations. Using analytics to gain insight into API usage should help in making future investment decisions. See the Lifecycle Management for more details.

To gather effective analytics, it is important for all consuming applications to have an API key in order to access the API. This allows driving out of values such as consumer location and associated peak periods etc.

5.1.10 State

It is important to understand where and where not to maintain state in the architecture. All effort should be made at keeping the API component completely stateless, meaning that every request to the API layer should be treated as a new request and therefore treated in the same way.

This does not prevent keeping information about the customer or the status of their request; it just means that state information is not held in the API component. This type of state information should be limited to the application or data layers.
5.1.11 API Design and Business Process Analysis

When performing the analysis required to design and develop an API, consideration should be given to all components in the reference architecture. This will ensure that the needs of all parties are met.

5.2 Expose Data as a Set of Resources

To identify resources, it is necessary to analyse the business domain and extract the nouns that are most relevant to business needs, or more accurately to the needs of the application developers and customers. The API needs to be relevant and useful from the perspective of API consuming application interactions. Once all the nouns, or resources, have been identified, then the interactions with the API can be modelled as HTTP verbs against these nouns. The HTTP verbs may not map nicely, so it may be necessary to approximate. For example, it is relatively easy to identify the nouns Customer, Address, Account, Teller, as resources in a banking domain. The aim is to offer a clean URL for each thing, and each collection of things.

5.3 API Publication and Access

The only way an external application developer can find and use your API is if the API is published. The ability to discover an API is fundamental to API take-up and innovation, and is one of the main requirements from commercial businesses. Any published API catalogue needs to be application developer-focused, as the primary point of discovery is the application developer. This means that an API must be well documented, and provide accurate and up-to-date guidance to application developer communities to encourage take-up.

Application developers need to be registered before they can develop applications which use the API. The process of registering an application against an API or API product should result in the issuance of access credentials.

Every API provider should offer a means of registration for application developers so that the consuming application base can be known and kept informed of events which impact access to APIs. All consuming applications should be registered with appropriate credentials as this enables API providers to understand their consuming application landscape.

Additionally, consuming applications that have been written to use an API will require a level of access relevant to that API or API resource.

- Where the API offers access to view public data, application developer registration should be automatically approved but still result in access credentials relevant to the application.
- Where the API offers access to view sensitive data, application developer registration should not be automatically approved and should instigate an agency defined due diligence workflow and will result in access credentials relevant to the application.
- Where the API is for development/testing purposes only, registration should be self-service and the ability for an application developer to "try out" the API should be provided. This is commonly known as "providing a sandbox"
- Where a customer is allowed to modify data, registration will include obtaining appropriate credentials for that customer, and an internal step within the API provider to define access controls for that customer commonly referred to as authorisation

5.4 Discovery

API catalogues will be emergent as more and more public sector APIs are published for use. Currently http://api.business.govt.nz/api/ offers an established catalogue of government APIs, but should by no means be the exclusive location, as private sector catalogues such as ProgrammableWeb and Mashape offer good API catalogue offerings.
When using a catalogue to identify an appropriate API, application developers are focused primarily on the resource they wish to access, rather than the government agency providing the resource. This is particularly relevant as governments and agencies change over time - the API product or resource should remain relatively stable. The business world tends to see government as one large entity holding valuable information, and are likely to know in detail what resource they are looking for but not necessarily which agency that resource would reside within. Hence it is important to expose APIs in terms of the resources they offer rather than the internal business process or agency structure.

The long term aim is for discovery of APIs to be automatic and dynamic, enabling real time integration of resources and onward presentation to customers.
6 Lifecycle Management

Developing and delivering APIs changes the model for most government organisations in that they are now creating and delivering products, in a very similar way to a commercial entity. With that comes a need to manage, maintain and support those products throughout their lifecycle in a professional way, otherwise take-up of the APIs will be limited and painful for all concerned. The following depicts the standard API lifecycle:

![API Lifecycle](image)

**Figure 8: The API Lifecycle**

Before an API goes live i.e. is released for use, provisions should be in place to:

- Support the internal development and testing of APIs
- Handle release management
- Support onboarding of application developers
- Define the level of service the application developer (and customer) can expect from the API (SLA)
- Support application developer usage of the API, including
  - Encompass Lifecycle/change management, handling API versioning and retirement management
  - Cater for Incident/Events
- Perform API Management

To some extent this mirrors standard service delivery practices defined in ITIL, but with an API-specific flavour.
6.1 Service Design

Service design covers the initial analysis and business drivers for the API as well as the way in which it will be built, where it will be hosted etc. Service Design not only applies to initial design of the API, but also to any changes and improvements over time.

6.1.1 Consumer Management

When designing an API, it is important to inform all potential application developers as to the thinking behind the API, including the resources to be exposed, the granularity of access, and capabilities to be offered. This can include roadmaps to indicate the planned evolution of an API so that developers are pre-warned and have time to prepare for changes.

Ideally the API services should be designed in collaboration with application developers to ensure the right product is developed to support diverse business processes across a broad set of consuming applications. It is best if API development can be performed in coordination with application developers who are producing consuming applications.

Identifying potential application developers and understanding the demand from the market can be difficult. Careful consideration should be given as to the best approach to engaging with the market and identifying your potential application developer community.
6.1.2 Service Level Management

Without robust service level management, it will be hard to engender trust in government APIs, which will negatively impact uptake. Application developers (especially commercial entities) will need to know how long the API will exist, what commitment there is to its availability and performance, and what support is offered to those who consume the API. Without this, API usage will be based on an untrusted model, where application developers prepare for the API being unavailable. This results in consuming applications using APIs to top up local caches of data, or to support existing batch processes, missing out on the real-time benefits of APIs.

Support could involve multiple platforms (some cloud-based, perhaps) and multiple vendors, and therefore the service levels offered in the SLA may be limited to that of the lowest common denominator.

6.1.3 API Design

This incorporates the tasks involved in the design of the API, prior to implementation:

- Business analysis of the target audience, the business processes potential application developers are trying to implement, and how the API fits into those business processes
- Agile processes for data analysis, resource identification, software design for development of the API
- Infrastructure planning, for the hosting of the API
  - It is worth recognising that for application developer support, an accessible test platform is just as important as the production platform which will host the final API
- Responding to requests for additions and modifications from application developers via Consumer Support (so Service Analysis does not just apply to initial design, but also to any changes)

6.2 Service Transition

Service transition focuses on the implementation build of the API and deploying the API for operational use.

6.2.1 Internal Development and Testing

APIs need to be developed in a collaborative, flexible and adaptive way. Once the API interface specification is agreed, it is more important to get something developed, however small, and get that published for application developers than to try to build a complete API product for release. It may well be that initial versions of the API have functionality stubbed out (resources which can be called but return canned responses) or missing, which is valid so long as the documentation indicates this. Support for this form of iterative development can be enabled through:

- Agile practices - Agile practices ideally suit this form of iterative development as they focus on developing small, incremental releases, 'failing fast' (finding out what's wrong early rather than too late) and frequent delivery of products
- Configuration Management - all the components which make up an instance of the API should be held within version control, so that it is possible to rebuild a previous version if necessary. This involves:
  - Version controlling API interface specifications
  - Version controlling the associated API code
  - Keeping track of dependencies (e.g. external libraries being used within the API code)
  - Making sure access policies for individual consumers are version controlled
  - Being able to provably recover all the elements of previous iterations of APIs and rebuild/redeploy if required
• Automated testing - To make the incremental release pattern efficient, it is advisable to develop automated tests in conjunction with the API code so that testing becomes an intrinsic part of the build process.

• Continuous Integration - Tools exist which make it possible to spawn automated build, testing and subsequent deployment whenever an update of the code is placed into configuration management.

6.2.2 API Catalogue

Before an API is released for application developers to access, the API’s description and interface specification should be published to an API catalogue. The API catalogue will contain a list of all the APIs offered, along with their interface specifications and guidance on how to gain access, and use the APIs, including the granularity of access control. This information needs to be up to date and accurate in order to be relied upon by API application developers. Some API hosting technologies can offer an automatic cataloguing capability (see section 8, Getting Started with APIs).

6.2.3 API Release Management

Release management is an important aspect of API transition. The aim with API development is to make small changes and release often.

The release management aspects include:

• Versioning - Informing application developers:
  o Where the interface specification has been changed, a major version release is required, with appropriate warning to application developers, scheduled deprecation of the previous major version and support for migration to the new version
  o A minor version change release is appropriate for backend changes that have little or no impact on the interface specification, and should have minimal impact on consumers

Hence it is important to know who your application developers are. When the API changes, the interface specification must also be changed to reflect the changes.

• Planning API rollout: Ensuring all the API Artefacts are rolled out effectively and on time to the correct platforms

• Emergency patches: Informing application developers as to the need, and schedule, for emergency patch rollout, and ensuring emergency patching does not impede consuming applications

Releases should be made to a test/development environment first.

6.2.4 Lifecycle/change management

As illustrated above, an API has its own lifecycle, and it needs to be managed through the whole of its lifecycle, from creation to destruction. This involves:

• Monitoring performance of the API to make sure it adheres to the SLA offered (e.g. availability, signs of increasing demand requiring horizontal scaling)

• Monitoring usage to make sure an API, or API version, is only retired when the majority of consuming applications have migrated off it

• Trimming APIs – removing capabilities which are unused, have never been used and are not likely to

• Ensuring the API roadmap is up to date and gives a good indication of when major changes are scheduled to be made to the API

6.3 Service Operation

The Service Operation part of the cycle looks at the actual delivery of the services to the service levels advertised. It handles management of, and access to, the API and any underlying applications, and looks after the infrastructure which underpins it. It also includes consumer support and incident management.
6.3.1 Access Management

The first interaction Service Operations is likely to have with APIs is the initial act of on-boarding application developers. Onboarding should provide everything the application developer needs in order to interact with the API, including access to a test environment running a representative copy of the API underpinned by test data, documentation and preferably examples to work with. Consideration may be needed as to whether all application developers can work with shared test data or if they need individual test data specific to their purposes. In their day-to-day development activities application developers will want to be able to test the API without making (computationally or potentially financially) costly calls out to third party services, so it may be beneficial to provide mock versions of those APIs specifically for testing purposes. For full system tests, however, application developers will want their applications to test the full flow including any third-party service, so an automated mechanism for that may need to be built.

Access management includes providing mechanisms through which application developers can apply for, and receive, permission and associated details to use the API i.e. external developers who want to build applications which make use of the API and applications which will ultimately be integrated with the API. It also covers managing access to the API, including specific, fine grained control for consuming applications. This allows operations to deploy access policies to ensure a consuming application’s access to the API is in line with agreed constraints.

6.3.2 Consumer Support

Application developers will need a variety of support mechanisms to aid their use of an API, including:

- Getting set up to use the API
- Understanding what to use during development
- Support for testing of their application in use of the API
- Reporting issues with the API

These mechanisms should include:

- Telephone support
- Support desk email address
- Online forum/support community (see also Section 8.1)

It could also include:

- Interactive real time support

It is useful to include indications of the level of support so that application developers know which form of support will most rapidly address their issue e.g. 9-5 telephone support, 24x7 community forum, response times to API failure reports.

There should also be support for handling requests for change, modifications or additions needed to the API. It should also be possible to capture and handle requests for new APIs.

---

6 So that dependencies on the API can be known, and usage monitored and adapted to, it is worth registering all consumers of the API. This also enables update/issue alerts sent to all application developers whose consuming applications are using the API, and ensures that version deprecation is managed without adversely impacting consumers.
6.3.3 Incident/Event Management

Incident and event management is geared around events picked up through monitoring, and unplanned incidents, and involves substantial amounts of communication. This includes:

- Monitoring the system as a whole to identify potential issues and pre-emptively apply mitigation e.g. throttling to counter potential DoS attacks
- Capturing reports of, then informing application developers of, unforeseen incidents which are currently causing a disruption of service
- Informing application developers of remediation of incidents, including resolution plans and predicted completion times
- Deploying temporary fixes, if necessary

6.3.4 API Management

APIs need to be managed as products, very similar to software products that commercial entities release. API management needs to be consistent across all the APIs the provider is publishing.

API management looks after availability of the API. This can involve throttling to make sure all consumers can get access to the API within the bounds of the SLA. It can also include quota management, whereby consuming applications are given limited access (e.g. a set number of calls per hour) to protect the API from abuse or overuse. It should be possible to use analytics (see section 6.4.1 Analytics) to assess whether throttling or quotas are needed.

For more information about capabilities to support this activity, see section 8.1, API Components.

6.4 Continual Service Improvement

It is not sufficient to release an API and let Service Operation keep it running. There is also a need to ensure that APIs are running optimally, and can adapt to changing demand over time.

6.4.1 Analytics

Capturing and analysing data about an API in operation will pull out information useful to adapting to changes in demand. It is therefore useful to gather analytical data around:

- Take-up metrics, end user analytics such as location
- Tracking API consumers, their registrations and API usage
- API performance – identifying most commonly used APIs calls so they can be made efficient
- Event behaviour (e.g. common patterns of behaviour)
- Trace & diagnostics data

From an API take-up/consumption perspective it is useful to capture who, where, when, how, how often, and what device type is being used. Analysis of this data can then be used to demonstrate ROI.

Performance metrics are also useful, such as error rate, throughput, response time, transaction speed, backend performance, cache performance. These values can help identify trends and bottlenecks.

6.4.2 API Availability

It is important not just to monitor and gather operational data about running APIs, but also to use that information to improve API offerings. API availability is of utmost importance to API consumers, so the API providers need to monitor usage and respond
dynamically to increases in demand. This requires transaction throughput monitoring to identify potential bottlenecks or overloaded APIs and to ramp up availability to meet demand.
7 Application of the Standard

The primary intent of this standard is not to constrain the use of APIs in government, but rather to give agencies and vendors some common, default guidance on API implementation and thereby accelerate the development of government APIs.

For this reason, most of the specific technical guidance is marked as “recommended” or “advisory” rather than mandatory. Exceptions tend to be in areas of security, authorisation and referenced standards compliance.

Nevertheless, the ICT partnership framework has directed that use of the standard as a whole should be included within the GEA-NZ standards reference and required for government agencies within the scope of the ICT Functional Leadership mandate7.

Agencies should consult the Government Enterprise Architecture Group and its API Working Group in the design of APIs that will be used across multiple government agencies, and must secure their approval for any significant departure from the standard.

For the initial version of this standard (v1.0), agencies are requested to provide feedback on the standard, its usability, effectiveness and any areas of ambiguity to the Government Enterprise Architecture team (gea@dia.govt.nz).

It is recognised that many sectors or industries will have existing APIs and associated standards, many of which are established and governed outside an agency’s control. In such cases interoperability will generally be the primary consideration, and so the industry standards will prevail. Nevertheless, agencies are encouraged to review their existing use of APIs against this standard and consider whether any discrepancies reflect material business risks.

Agencies that have specified their own existing APIs will likewise need to balance the risk and cost of change against the benefits of conformance with this standard. A phased conformance may be appropriate.

8 Getting Started with APIs

When getting started with API development and delivery the following approach is recommended:

- Start small - pick one aspect or API product from the agency or information source and build a well-defined API.
- Work with a single application developer to ensure the interface specification meets their needs whilst still having relevance to broader use (i.e. not aligned solely to their business objectives)
- You will need a few basic capabilities set up to support development, discovery of and access to your API:
  - An Application Developer Support Environment – Developer Portal tools can aid in this
  - API Gateway – for ensuring safe access to your APIs
  - Some means of managing published APIs – an API Manager can assist in this
- For development and hosting of the API, consider using off-the-shelf (OTS) products or software as a service (SaaS) which is subscription-based (e.g. based on number of transactions) to reduce maintenance and support costs. Subscription based offerings allow an agency to start with a small number of transactions and increase these over time as their API ecosystem grows.

---

7Ref CAB Min (12) 35/4, SEC Min (14) 5/2
• Use analytics to gain visibility of take-up, developer engagement and usage profiles, and adapt APIs to meet changing needs over time

8.1 API Components

The following are definitions of components which provide the development support capabilities listed above:

![Diagram of API Core Components & Capabilities]

**Figure 10: API Core Components & Capabilities**

8.1.1 API Developer Portal

Agencies developing APIs need to be able to engage, onboard, educate and manage consuming application developers whether inside or outside the organisation. These management activities will generally include registration, documentation, analytics etc. The easiest way of making these capabilities available to application developers is to offer a dedicated web site – or development portal – which should offer:

- **Discovery** – Making it simple for developers to find APIs which match their need
- **Onboarding** – Allowing developers to register and sign-up for API usage plans with support for automatic approval or manual approval workflows
- **Education** – Providing developers with the information they need to make use of APIs
- **Examples** – Offering example code and sample applications to illustrate the functionality of the API
- **API Evaluation** – Giving developers a “sandpit” capability to try your APIs interactively
- **Community** – Enabling developers to share knowledge, experience and best practices via a support community such as online forums
- **Metrics** – Delivering insight into API usage and performance which could be useful to developers
- **SLAs** – A place to publish API SLAs, defining performance, availability metrics etc.
8.1.2 API Gateway

The API Gateway is the means through which APIs are offered to the outside world. This component (physical or virtual) hosts the APIs ready for consumers to access. It provides an agency with the ability to control who has access to their APIs by enforcing policies for access control. The gateway is sometimes referred to as the API policy enforcement point. Some API gateways also offer additional capabilities such as:

- **Security** – Offering authentication and authorisation services. Using threat protection capability to protect backend systems against common vector attacks such as SQL injection, Cross Site Scripting and Cross Site Forcery. The API Gateway is essentially the Policy Enforcement Point (PEP)

- **Performance** – Maximizing API efficiency to support consuming applications and minimizing downtime

- **Data transformation** – Converting transiting data into application-friendly formats

- **Orchestration** – Composing new, aggregate APIs from multiple existing APIs

- **Quality of Service (QoS)** – Provides QoS for consuming applications and enables an agency to apply policies for a specific consuming application or policies that protect back end systems from overload. This is commonly referred to as “throttling” and “quota”.

- **Logging** – Saving events as messages in a file or database for later analysis and auditing

8.1.3 API Manager

The API Manager is a technical layer enabling an agency to control an API’s visibility and behaviour. It is usually exposed as a UI/console to internal staff only, as it is seen as an administration component. It offers:

- **Access Control** – API manager is the place where API administrators can control API access such as defining access policy for the API gateway to enforce, disabling an API key or other credential that is being used maliciously.

- **QoS Policy Definition** – Administrators use the API management layer to define QoS policies such as defining quota limits for an application or API.

- **API Registration** – Administrators register APIs in the API manager so that they can be exposed to application developers. API developer portals generally communicate with an API management layer to obtain information about exposed APIs and to push application developer details. As part of the release process API administrators register APIs in a management layer.

- **API Catalogue Administration** – Providing governance over the API catalogue, what is exposed and what is not, what is available for external use or internal use only

- **Lifecycle Management** – Managing the full lifecycle of APIs from initial publication to destruction

The main capabilities usually support:

- Planning and design
- Implementation
- Basic and advanced deployment and running
- Versioning and retirement
- Allowing policies to be applied to individual APIs
- Helping to onboard consuming applications

See Part B, API Security for technical details about implementing security.

9 API Governance

This version of the API standard does not yet offer detailed guidance on API governance.
## 10 Appendix: Glossary

Below is a list of common terms. Terms in the Description column highlighted in bold are described elsewhere in the Glossary.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytics</td>
<td>Analytics in the context of this document is the capturing and reporting of API usage.</td>
</tr>
<tr>
<td>Consumers</td>
<td><strong>Customers, API Consuming Applications</strong> and <strong>Application Developers</strong> who use the <strong>API</strong>.</td>
</tr>
<tr>
<td>Consuming Application</td>
<td>This is any application (on any device) that consumes an <strong>API</strong>.</td>
</tr>
<tr>
<td>Context</td>
<td>Context in this document generally refers to request context. For example a <strong>JWT Token</strong> may contain information about the customer initiating an API request.</td>
</tr>
<tr>
<td>Customers</td>
<td>People (or organisations) that use the <strong>Consuming Applications</strong> to access the <strong>API</strong> resources the <strong>API Provider</strong> is offering.</td>
</tr>
<tr>
<td>Discovery</td>
<td>The ability for <strong>Application Developers</strong> to find resources and associated APIs to use in their <strong>Consuming Applications</strong>.</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface - a piece of software, which provides a way for other disparate pieces of software (applications, systems) to talk to one another.</td>
</tr>
<tr>
<td>API Catalogue</td>
<td>The <strong>API</strong> delivery component that lists the <strong>APIs</strong> offered, along with their <strong>Interface Specifications</strong> and guidance on how to gain access and use the <strong>APIs</strong>.</td>
</tr>
<tr>
<td>API Developer</td>
<td>The organisation (or person) who creates the <strong>API</strong> and maintains the <strong>Interface Specification</strong> for the <strong>API</strong>.</td>
</tr>
<tr>
<td>API Developer Portal</td>
<td>The <strong>API</strong> delivery component that allows <strong>API Providers</strong> to engage with, onboard, educate and manage <strong>Application Developers</strong> whether inside or outside the organisation. These management activities will generally include registration, documentation, analytics etc.</td>
</tr>
<tr>
<td>API Gateway</td>
<td>The <strong>API</strong> delivery component that allows <strong>API Providers</strong> to offer <strong>APIs</strong> to the outside world. This component (physical or virtual) hosts the APIs ready for <strong>Consumers</strong> to access. It provides an <strong>API Provider</strong> with the ability to control who has access to their APIs by enforcing policies for access control. Some API gateways also offer additional capabilities.</td>
</tr>
<tr>
<td><strong>API Manager</strong></td>
<td>The API delivery component that allows API Providers to control an API’s visibility and behaviour. It is usually exposed as a UI/console to internal staff only, as it is seen as an administration component. It offers a variety of capabilities, including API registration and catalogue administration.</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>API Provider</strong></td>
<td>Organisation who provides the API to expose a resource (information or functionality) for Consumers.</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>The software behind the API which provides the business logic for the resource.</td>
</tr>
<tr>
<td><strong>Application Developer</strong></td>
<td>Software developer or organisation who builds Consuming Applications that use the API. An application developer can be internal to your agency, developers that work with trusted partners, developers from other agencies or developers from the private sector.</td>
</tr>
<tr>
<td><strong>Interface Specification</strong></td>
<td>Provides technical information to the API Developer community about the API. It includes information about how the API works, any access control features and any error conditions.</td>
</tr>
<tr>
<td><strong>Product Manager</strong></td>
<td>The product manager is usually a technical role. They understand an agencies API landscape and are owners of API management platforms.</td>
</tr>
<tr>
<td><strong>Product Owners</strong></td>
<td>The product ownership function usually resides in a business area rather than technology. The role of the product owner is to understand the product that the agency is trying to deliver and be able to make decisions on the representation of the product in an API.</td>
</tr>
<tr>
<td><strong>Publish</strong></td>
<td>The act of releasing the interface specification and associated API to a location accessible by Application Developers</td>
</tr>
<tr>
<td><strong>Resource</strong></td>
<td>The information or functionality exposed by the API.</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>State defines the point in time record of an in-flight transaction. Some systems maintain user state for a period of time to enable a transaction to be continued from the point of last recorded state. APIs are usually, but not always, considered stateless.</td>
</tr>
</tbody>
</table>