# Stimulus

## Technical specifications

### Identification

The Australian Government is interested in funding development of an application programming interface (API) for its travel advice resources. The API will be used for Australian Government employees to manipulate travel advice, and for third-party application developers to seamlessly interface with, regardless of technology stack.

A proof of concept solution is required that demonstrates:

* an API implementation
* a web application that simulates public and private data exchanges with the API

The API implementation will:

* organise travel advice into a logical data structure or store
* provide endpoint URLs that can:
	+ view data publicly
	+ manipulate existing data (preferably with secure authentication)
* document endpoint URLs
* return resource requests in a ubiquitous syntactical format (E.g.: JSON or XML)

The web application will:

* request data from the API implementation using documented endpoints (a public access key is not required for requests that do not modify data)
* accept client-side values via HTML forms, and use these to filter requests sent to the API
* appropriately format data from API responses and deliver the resulting mark-up to the client’s browser. Rendered HTML should conform with industry standard accessibility guidelines
* supply a private key to the API when making requests from the web application to modify existing API data
* provide a layer of security beyond public access areas. Private API keys should only be delivered or accessed via this secured area of the web application environment.

In summary, the proof of concept solution involves:

* developing an API and web application that simulates public and private data exchanges for the Australian Government Smart Traveller advice
* generating the components that simulate the exchange of data between the API and web application, using programming techniques to transport data in an interchanging format
* evaluating impacts and making recommendations for improving data security during transfer
* developing a video to demonstrate data transfer functionality

### Interactions

Proto-personas have been developed for potential users of the API and / or web application:

**Figure 1: User profiles for the new web application**

|  |  |
| --- | --- |
|  | Andrew* Foreign journalist who enjoys backpacking in his spare time. Uses Chrome to check his emails and subscribes to RSS feed on Smart Traveller
* Likes to travel to countries that are considered “High degree of caution” level of danger
* Knows that Australian consulate has limited or no support in certain locations, and as such, will not travel to “Do not travel” / highest level of danger
* Limited computer skills and doesn’t travel with a laptop. Makes use of internet cafés, which are quite often slow, old desktop systems in poorer countries
 |
|  | Chantelle* Mobile application developer
* Working on a “Digital Passport” application, that securely stores sensitive documents (such as a passport or VISA) on a traveller’s device, as well as providing alerts on changing travel advice
* Seeking an API that can be periodically queried and deliver a light-weight response that can be easily stored on a traveller’s device during extended periods of intermittent internet access
 |
|  | Wendy and Gavin* Smart Traveller employees providing travel advice
* Are required to be on call every day of the year, as travel advice can change very rapidly
* Work from office computers and home laptops to keep travel information up to date
* Very strict on security
 |

### Component Specifications

#### Data

The API implementation must organise travel advice into an appropriate data structure for public access and private manipulation. As this is a proof of concept solution, the travel advice should be limited to:

* + two of the four regions with sub-regions available (Africa, Americas, Asia or Europe)
	+ one of the two regions without sub-regions available (Middle East or Pacific)
	+ twelve countries (in total). The twelve countries chosen should spread across each of the three chosen regions, ensuring that each region has at least one country, and that all the following levels of danger are represented at least twice:

|  |
| --- |
| Level 1 - Exercise normal safety precautionsLevel 2 - Exercise a high degree of cautionLevel 3 - Reconsider your need to travelLevel 4 - Do not travel |

* + appropriate data for this API implementation to store, which includes:
	1. Country name, region and sub-region (if it exists)
	2. Level of danger (E.g.: Level 4) and description of danger (E.g.: Do not travel)
	3. Issue date (E.g.: 28 May 2019)
	4. Travel advice (limit to one paragraph, usually the first appearing on the page)
	5. A URL web link to the country page located on the Smart Traveller website (E.g.: <https://smartraveller.gov.au/Countries/pacific/Pages/new_zealand.aspx>)

Functions requiring data in the web application may include:

* requesting filters or parameters for API requests, such as region, danger level, issue date, or substring search within an advice summary (E.g.: “riots”)
* supplying an API key for modifying existing data within the API. An API key may be a 128-bit UUID, that is supplied to the API as an argument in an endpoint URL. This key should only be accessible in a secured part of the web application. If a key supplied is valid, then appropriate data modifications (such as changing levels of danger, or travel advice) could be made to the API data via path segments or parameter values supplied in the URL
* securing authentication for credentialed users to access pages outside of public web access. It is suggested that these credentials are stored via a session variable. Any passwords or private access API keys should be stored using a secure hash algorithm

#### User interface / experience

The API must provide documented endpoint usage. The web application may provide:

* browser independence, be responsive and / or adaptive, and conform to web usability principles
* views for public and secure access of the API, using HTML forms and client-side validation for input, and rendering dynamic HTML results in a readable, organised format
* suitable timeout or disallowed authentication messages on failure of requested resources
* client-side validation of forms

The UX should be considered in planning and incorporate web usability principles.

#### Code

The proof of concept solution must contain accurate and efficient code, that is planned using detailed algorithms illustrating how requests such as searching, sorting or filtering data are to be fulfilled. General purpose algorithms that enable a secure data exchange between client, web application and API must also be considered.

The API must:

* be able to parse a valid URL for path segments, query string parameters and / or parameter values
* return a valid response in the required format if a request for public data is made
* organise end points with semantic meaning. URL path segments should be short and descriptive and utilise a natural hierarchy of path structure. Be careful and consistent with pluralisation

The API may also check if a supplied private access key is valid using a secure hash algorithm, and if so, manipulate data as requested.

The web application must:

* request user inputs via HTML form elements, and use appropriate HTTP methods (E.g.: POST or GET) to transmit these values between client and web application server for requests or operations to the API
* ensure valid public and private access through client authentication strategies
* perform API requests, and re-encode the response from the interchangeable format that is returned into a format that can be manipulated programmatically (such as a dictionary, list or tuple)
* generate mark-up that delivers API responses to a client browser

### References

* Australian Government 2019, Smart Traveller,
<https://smartraveller.gov.au/> accessed 14 July 2019
* Australian Government 2019, Smart Traveller – RSS feeds,
<https://smartraveller.gov.au/resources/Pages/rss.aspx> accessed 14 July 2019