pAPI

The Public API you never asked for!

ABSTRACT

APIs available where you are, when you want them, and how you want to use them

Poni | Oliver | Kurtis | Waseem

Software Development

Table of Contents

IT721
Software Engineering
Final Project

SOUTHERN
INSTITUTE OF TECHNOLOGY
TE WHARE WANANGA O MURIHIKU

Please sign the following statement: “I declare that this assignment submission will be my own work and I will not collude with anyone else on the preparation of this Assignment.”

Name / Student ID:

Date:

Content	**Mark**	**Result**
System	35	
Debugging and Testing	10	
Implementation	5	
Presentation	10	
Sum	**60**	

Introduction

An API is an acronym for Application Programming Interface, which in laypersons' terms is an intermediary that allows applications to "talk" to each other. Every time you use any application for messaging, checking sports results, or popular applications like Instagram, Twitter, or Facebook you are using and API. APIs add a level of extra security, providing a buffer between a server and a client. The Modern API has become a valuable key that not only adheres to standards (HTTP and REST) they are developer friendly, accessible, and easily understood.

pAPI (Public API) will assist the user to find, connect to, apply, and manage APIs across multiple environments through an API hub/Library. Each API has a structure, data format and networking protocols which are unique, finding the right one can be difficult and time consuming. ***pAPI*** will be developed so that the user can browse by category, type by using a drop-down menu, a search bar, or theme tag that will group the APIs based on similar or complimentary functionality.

pAPI is designed to integrate any API into your application, with an easy to use and understand user interface (UI) and with a uniquely easy user experience (UX).

pAPI will bridge the developers, and users' gap for accessible API integration and use. The professional and the novice alike will be able to search for, view, and apply relevant APIs to their applications as easy as one would choose a song on Spotify, or a video on YouTube. ***pAPI*** is the future of personal, corporate, or professional us of APIs.

(Khan Saad Bin Hasan, 2020)

Purpose

To provide a web Application/Site that has a collection of useful APIs that can be searched, browsed, and utilised by users.

Users:

Those who will primarily benefit from the new web application system.

Customers:

Upon discovery of the web application, customers will find the site, use it for navigation, API identification by category or type filtering and through a search feature, and product display for use. Customers will be able to browse, investigate, instantly use snippets, or be directed to the relevant source material for application implementation.

API owners:

The source for the API will be given credit and relevant links to their products for implementation, or recognition which will provide a localised source of advertising and product recognition.

Intended Audience

This document will provide for any user, developer, tester, and project manager the basic system architecture and systems specifications.

Potential Users:

Developer

The developer can assess requirements, add, or subtract new or existing requirements and provide updates to the documentation via updates protocol. This document can be used to clarify position, progress, and direction of the system architecture.

User

The user can assess and critique the understanding of the deliverables, and system requirements as an advocate for the end user

Tester

The validation testing can be derived from this document so the tester can clarify the executable programme.

Scope Overview:

The objective of ***pAPI*** is to facilitate a platform system for users to access API data relevant to their interests. ***pAPI*** will provide a gateway for users to access APIs to research, discover or use in their applications.

pAPI is a complete web application that will provide and achieve the following:

- A landing page that provides a comprehensive dashboard for searching, browsing, and listing API by name, function, ad theme.
- Provide access to various useable APIs without traversing multiple websites.
- A function to choose, search for or discover APIs in a search bar, drop-down, or via grouped theme tags.
- Browse basic information about Latency, success rates, and popularity of particular APIs.
- Provide a more detailed profile of individual APIs with information about how to implement code, how to subscribe for immediate use, basic data structure, documentation, and functionality.
- A cached memory of API information for reliability and maintainability.
- ***pAPI*** will provide an access to a hub of APIs that can be sorted and categorised, and readily available for implementation into any application the user desires.

Critical dependencies & Assumptions:

- All software and hardware will be available for use. (Including Servers).
- Project approval is provided through the participation and availability of all stakeholders in creative and technical reviews.
- The API keys will be capped to restrict overuse.
- All APIs that are used will be free to the public and do not require paid subscription.
- Developments in technology will be addressed as and when required.
- All personnel will be available and attend all team briefings, meetings, and will participate in all aspects of the design, development, and implementation.

Constraints:

Software builds will provide a range of challenges that can be identified as critical to a successful development. ***pAPI*** understands and recognises that there are conditions that exist outside the control and influence of the pAPI developers and limit the design alternatives available to the project team.

- Local, and international standards. Where possible pAPI follow the standards established by IEEE, ISO and in New Zealand MBIE, IRD, and MIA.
- The timing schedule is approximately 17 weeks. (Teams had not been finalised until week five, and COVID19 precautions has meant online classes, and non-contact time).
- pAPI has no budget for development, and hence server costs, testing, and development time must fall within the means of the project developers.
- Technical aspects such as software, and development tools are limited to what each individual has available.
- Security features may be limited by SIT firewalls, protocols, or structure.
- Further Health and safety risks due to COVID19 conventions.
- SIT policy, and regulation.

Scope exclusions:

- Photo assets production and image postproduction services
- Licensing and hardware costs
- Liabilities for third party service partners
- Hosting, font, or service fee
- Support for other operating systems, browsers not directly outlined.

User

Any user that is not identified as a ***pAPI*** developer. A user may search for public information by Keyword, category, Tag, functionality, or theme. A user will be able to access whitepapers, synopses, or landing page displays of available APIs.

API Developer

The owner or manager of a specific API. The owner receives calls from the ***pAPI*** server for API data and documentation for display for the user. The owner may update, delete, edit, grant, or restrict access to the API by ***pAPI***.
pAPI admin/Developers
The key developers of the ***pAPI*** system and web Application

Current System

Assumptions:

- The API market is unlimited, and growth will remain the same as the beginning of the project.
- No major Market changes will affect the production of APIs and their usefulness in the market.
- The existing system is targeted solely towards the use of APIs by industry professionals
- There is no existing completely free API website for developers or Laymen users.
- Full use of Free API functionality will be available to pAPI
- All stakeholders have been consulted and responses recorded.
- The requirements and specifications document will represent the needs and wants of the stakeholders
- Use an API in this context is being able to implement a snippet into a program without downloading or acquiring and individual API key from the developer.
- All the free public APIs have been acquired through the correct channels and have been granted the full license for use on their behalf.
- Any database caching will be in accordance with all relevant privacy and storage rules dictated by the API developers, and pAPI.
- The databases will only hold information until the next update.
- Combining the basic functionality of the two examples is representative of the whole industry

The current system or existing systems (similar websites) is a comparison between two of the existing websites that offer something like pAPI.

These examples are RapidAPI and API List, both websites offer a hub/library of useful APIs and add value by providing access to snippets and developers links for licensing and API keys for further investigation or use.

API list [...]

API List launched in 2017 and quickly became one of the most recommended API resources on the web. It is visited by 100s of thousands of developers a year and ***viewed over 70k times a month*** *(and growing). It is also* ***referred to by many computer science universities*** *around the world as a valuable resource when teaching about APIs (API List, 2019)*

API List offers a window into the world of APIs. It has a comprehensive catalogue of useful and interesting APIs. This site offers glimpses into what the APIs are and a link to the developers. If you are a developer, it acts like Ikea and shows you the basic information about what the API is and can do. It will then take the trained eye to the next stage where applying the API can begin.

This system is simple, where it is easy to

RapidAPI

It all began at a hackathon in 2015. RapidAPI was made by developers for developers so they could have one place to access APIs and Microservices and build applications more efficiently and easily. Today, RapidAPI is the world's largest API Hub where almost three million developers can find, test, and connect to tens of thousands of APIs — all with a single account, single API key, and single SDK.

Software developers can also share and collaborate on internal APIs using RapidAPI for Teams, a common workspace to publish internal APIs and share public API subscriptions. In turn, organizations can use RapidAPI Enterprise to create a centralized hub environment to help developers reuse and connect to existing APIs faster while providing IT with enterprise-wide visibility and governance of API consumption. (RapdiAPI, 2022)

RapidAPI is a comprehensive system, which requires a membership to get a real use for the website. Although you can browse the information in a test capacity you are required to register and sign in to receive all the benefits of the site. There is a broad range of resources available on the site from blogs to articles about APIs and much more. RapidAPI works mutually for the potential developer and the established developer, RapidAPI has created an environment of enterprise, innovation, and cooperation.

Both these web sites allow a user to engage the web address and simply browse what APIs that are available and their simple information. Both require a membership or login in order tom use wider functions. Both RapidAPI and API list provide an API search feature, and both are aimed at the established developer or knowledgeable user.

The following Ishikawa diagram demonstrates the limitations of the existing systems. They demonstrate the frustrations users demonstrate in the lack of engagement with these sites.

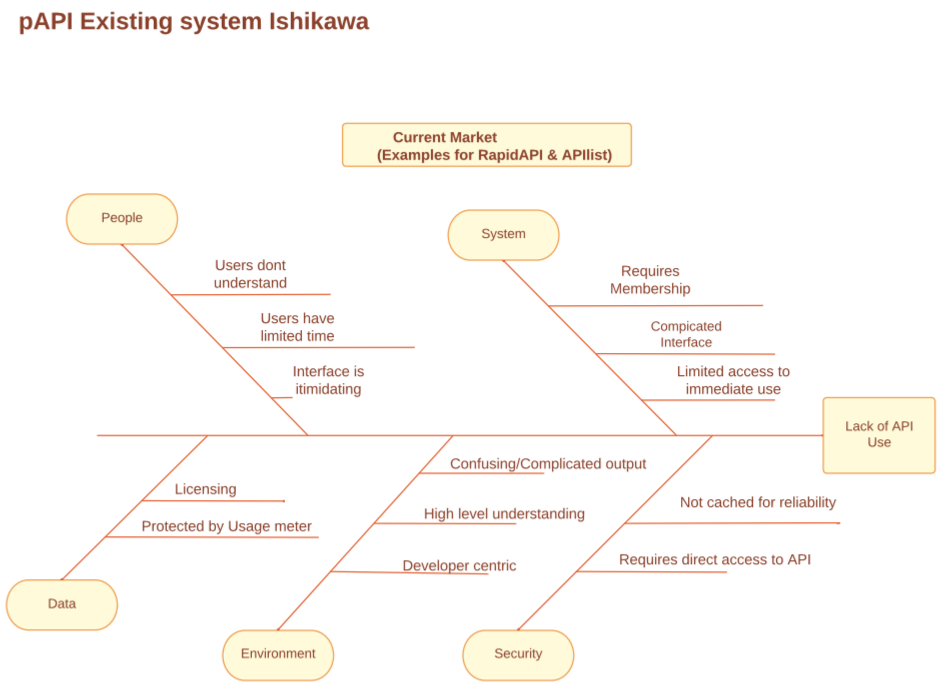


Figure 1 Ishikawa

Proposed System

System Features:

Description and Priority

The ***pAPI*** system maintains a hub of APIs of various uses, it provides basic information about the API, its applications, and implementation within the user’s application. The modern API is a pathway to successful deployment of mobile applications, web applications, and user systems.

The basic outline of the functional requirements is broken down to responsibilities of users, developers, and the server system. These are :

User Requirements

- Use snippets of code to access information they require
- Use link to access API themselves
- Utilise system formatting for personal use

Developer requirements

- UI UX (REACT)
- Search API Titles
- Thumbnail images
- Whitepaper of API functions, usage.
- View API formatted information

Server Requirements

- Find and use API's
- Create database to cache API data for maintainability
- Refresh data regularly for latest version of data
- Store JSON information for use on customer interface
- Deliver useable JSON for UI

System requirements:

- System properties
- Response time
- Storage
- I/O device capability
- System representations

Functional priority Objectives

High Priority

1. The application shall facilitate for API acquisition, inspection, and utilisation by the user. For users this will eliminate the need for a self-search of Developer or vendor websites and provide an instantaneous snippet for immediate use or a direct link to the developer for larger implementation, subscription, or licensing.
2. The application shall reflect a new and changed product description at each database update by a key call from the ***pAPI*** server and secured in the MongoDB database. This will keep API data relevant and current.

3. This application shall display information that is customized based on the users' requirements. Creation of multiple styled filters by type, functionality, theme, genre, and title. This will direct the user to the most valuable APIs for their needs.
4. This application shall allow users to view any and all available APIs that are available in the hub.
5. The application shall allow a customer to directly contact the API developer.
6. This application shall provide accurate and maintainable API data for immediate use and implementation via snippets.
7. This application shall provide descriptions, images, and documentation that will give a synopsis of the API and its uses, and applications.
8. This Application will provide in depth , accurate, and up to date information about the API so the user can make informed decisions about implementation of the API.

Medium Priority

1. The system shall provide a search/browse facility of all available and relevant APIs.
    - full text searching,
    - theme,
    - category,
    - type,
    - functionality
    - tag searches
2. This application will make whitepapers available from the API page. This will provide answers to user questions and reduce customer support.

Low Priority

1. This Application shall provide translation services for different users. Saving support calls.
2. This application shall provide metrics that will tailor the displays to the most popular APIs first. Using usage data to adjust home page displays.

Non-Functional Objectives

Reliability

The system shall be operational at least 95% of the time.
Down time after a system failure should be less than 60 mins.

Usability

A user who knows what they want should be able to locate and view that API in 10 seconds.
The number of pages required to be navigated to access product information should not exceed three.

Performance

The system should be able to support and maintain one hundred users simultaneously.
The mean time to view a web page should be not in excess of 10 seconds.
The mean time to view a white paper on the API should not exceed 10 seconds.

Security

The system shall provide a level of security so that users cannot exceed the max calls to the API developers
The system shall not record any information about the user except for metrics on popular usage.

Supportability

The system should be able to provide access to new APIs and developers without re-engineering the entire application.
The system web site will be available and viewable on Internet explorer 5.o and above, Netscape 5.0 and above, google chrome, safari, and Mozilla Firefox.

Online User Help

The website will incorporate a user guide/Walkthrough that can be opted in or out of.
The help bot will be available from all pages.

Cost Components

A web site search engine will be required
A language translation tool should be available.

Interfaces

The system must be able to interface with all popular developer platforms.

Need

This system will provide an easily accessed and instantly usable API hub for savvy or novice customers, guests, or users.
At present APIs are the sole domain of developers who have an affinity and knowledge of their potential implementation. ***pAPI*** will bridge the gap between developers and the everyday computer user who may just enjoy

games or are beginner programmers. This will be the introduction of these characters to the world of API implementation in innovative and new arenas.

Objectives & Success Criteria

<u>Objectives</u>

Decision Support
Provide up-to-date and relevant information and access to free and useful APIs from a variety of developers.

Customer experience
Generate increased customer engagement and use of developer's APIs.

Productivity
Provide access and use of APIs that will enhance the productivity of developers by providing usable access to snippets and links to API developers

Capabilities
Save a business or developer the time and energy to store a library of their own by providing full access to a vast library.

Knowledge
Improving competitive intelligence and access to relevant APIs

Data
Improving data quality

Integration
Creating workflow directly with relevant APIs

Skills
Teaching developers' new skills and keys to access more specialized APIs

Performance
Improve performance of the developers by lowering the time taken in the search and use of relevant APIs

<u>Success Criteria</u>

Cost
The costs of the project do not exceed the budget for development.

Timeline
The project is completed prior to the deadline provided by the client

Scope
All aspects of the scope are completed

Deliverables
All deliverables are completed on time and presented for approval by the team leader and client.

Resource capacity
The relevant technology stack is utilized and deployed without external influence.

Stakeholder Satisfaction
The stake holder provides a grade representative of 80% or more.

System Models

Scenarios

The basic user journey dictates the functionality of the user interface.

In analysis of the proposed system, we should consider "how" and "why" the system will be used. Of particular concern of the is the user. WE define the user in terms of any actor that would download the website. This definition does not preclude the experience of the user, and so the user story in a graphical sense should encompass the generic role of the user.

The following diagram is a simple User Journey diagram that outlines the search/browse options the user would initiate. As demonstrated in the diagram the user has a binary decision on their choice to engage with the website and search or browse the pAPI application. This leads to a process that activates the procedures and protocols within the application.

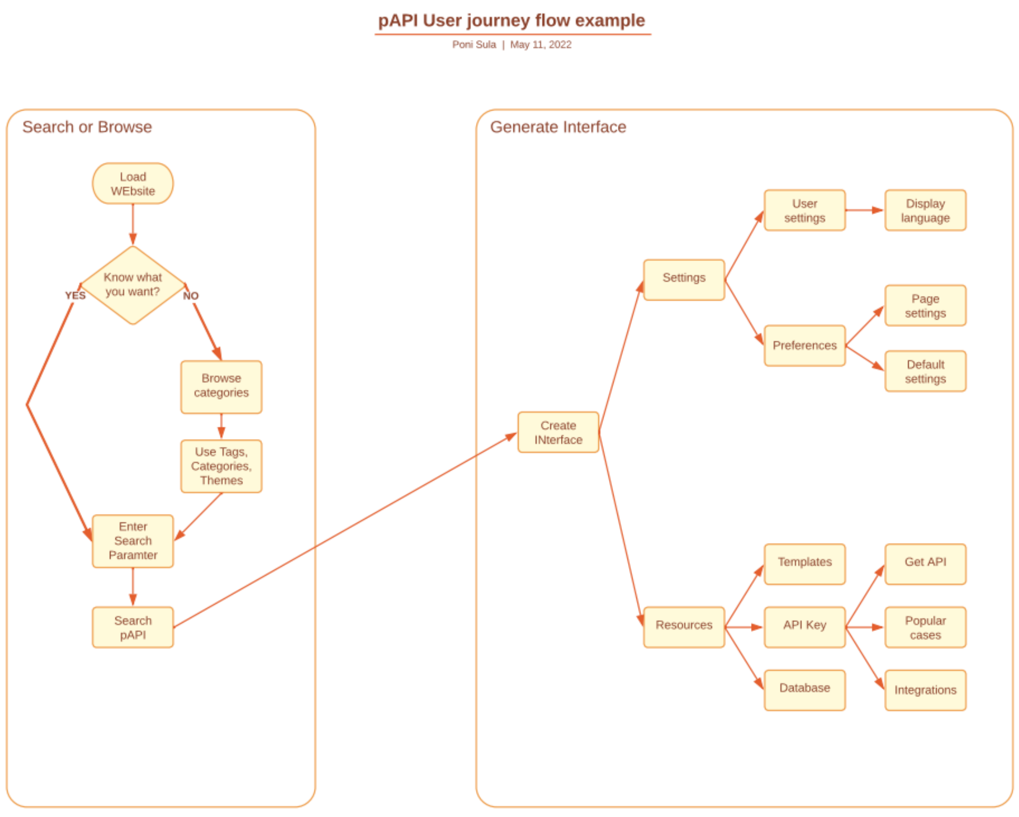


Figure 2 User Journey Generic pAPI

Use Case Model

For more specificity we have generated more direct user stories.

User Story one

Title:

As an experienced software developer, I want to add a weather API to my app so that my users can check the weather while using my app.

Description:

The user is an experienced mobile developer building an app which utilizes current weather data but does not currently have a way of collecting that data him/herself.

Acceptance Criteria:

Provide multiple examples for implementation with a variety of popular programming languages.

Search functionality.

- *Give an example of what data will be returned from the API.*
- *Minimal number of clicks to get the required information.*
- *Provide detailed documentation.*

User Story two

Title:

As a beginner programmer

I want to try make my first ever API call

so that I can further my programming skills.

Description:

This user is a beginner programmer who has never worked with API's before and wants to write a basic program that displays the output of their first ever API call.

Acceptance Criteria:

- *Easy to follow instructions.*
- *Provide multiple examples for implementation with a variety of popular programming languages.*

User Story three

Title:

As a software developer building my first commercial app

I want to compare different APIs to see which would fit my client's needs best

so that my client receives the best product possible

Description:

The user is creating their first commercial app for a company and wants to make sure that the API they are choosing is the best option for their clients' needs to provide them with the best product possible.

Acceptance Criteria:

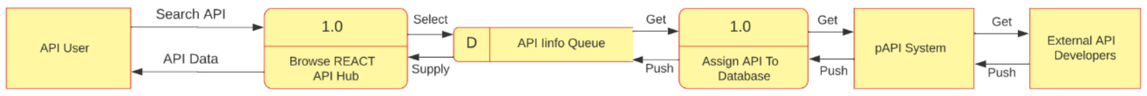
- *API categories.*
- *Shows basic metrics of each API.*
- *Provide multiple examples for implementation with a variety of popular programming languages.*

The use cases are as follows.

Analysis Object Model

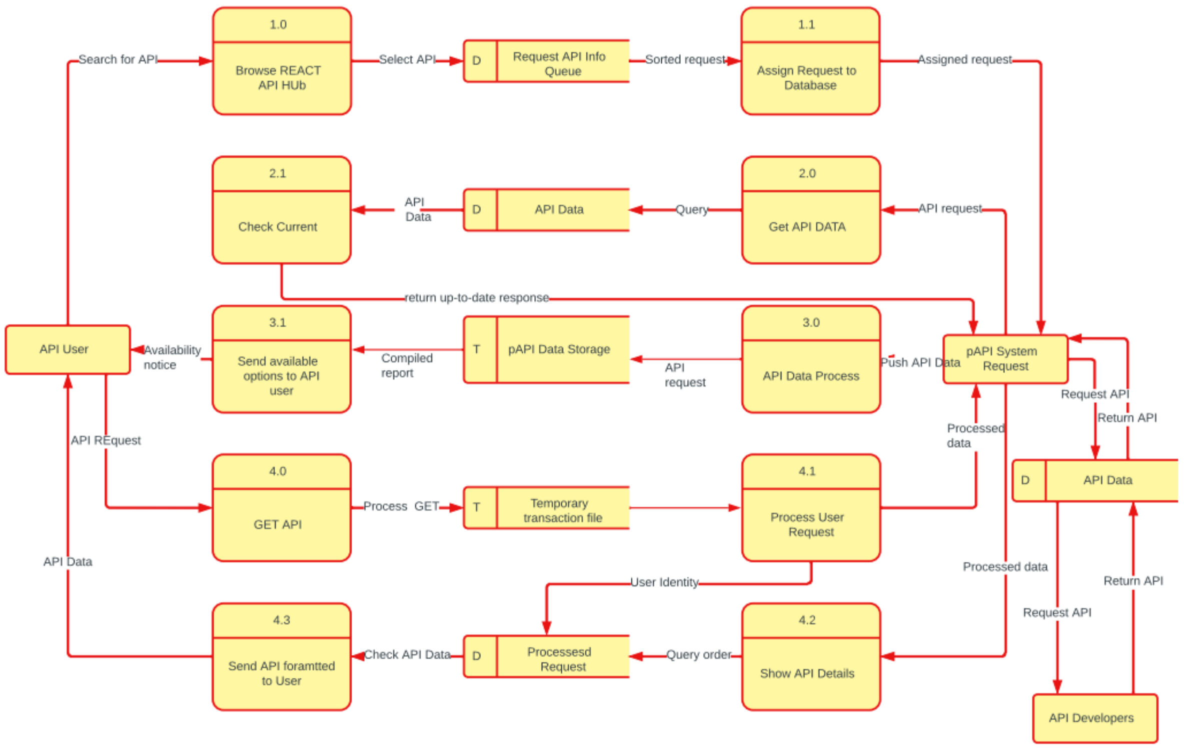
Data Flow Diagram CONTEXT (Physical)

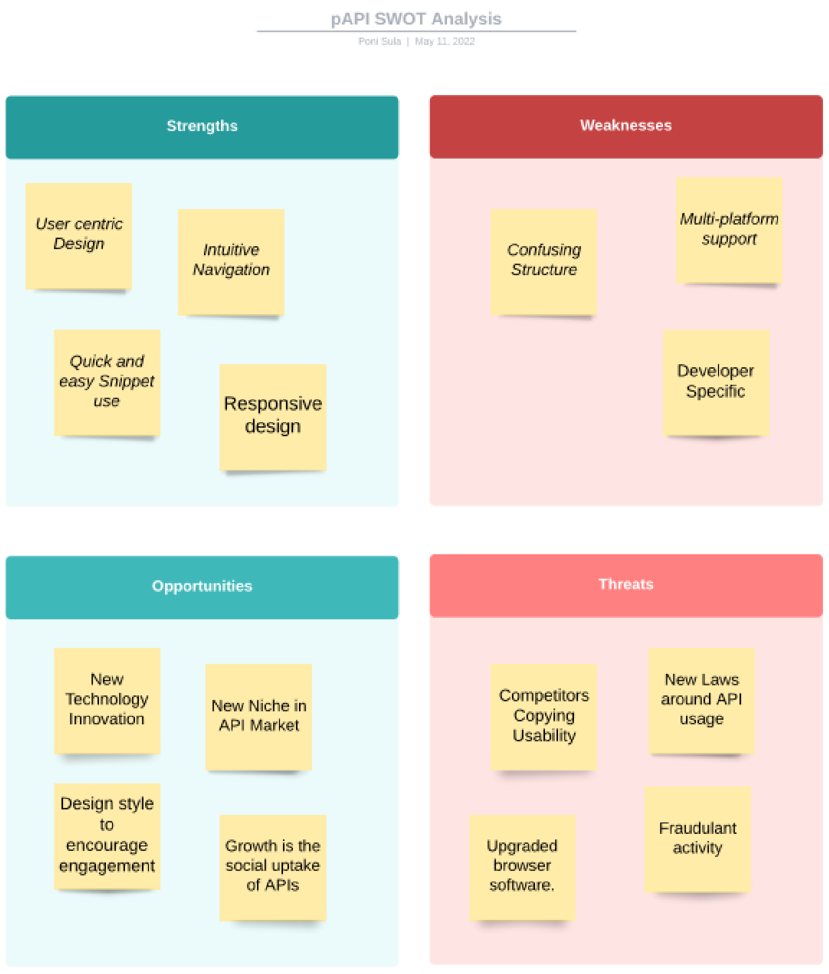
Poni Sula | May 11, 2022



Data Flow Diagram (Physical) Extension

Poni Sula | May 11, 2022





Dynamic Model

User Interface Navigational Paths

Design Specifications

Document Organisation

This SDD describes entirely the system at the architectural level, including subsystems and their services, hardware mapping, data management, access control, software structure and some boundary conditions. The SDD is organised into sections, each section provides detailed subsections relevant to the heading section. Charts, tables, and graphics have been provided to explain and clarify the content. Some of the subheadings reiterate themes, this is to ensure all perspectives as pertaining to the heading have been covered comprehensively.

System Overview

pAPI Project team is part of a cooperative agreement with Abdul Rahman (Client) to create system specifications for a web-based application that will provide access to and use of public APIs.

The **pAPI** system, referred to simply as **pAPI**, will call APIs from developers, store these in a NoSQL document-based database, and then allow the UI to format and display them for consumption by the user. To achieve these goals, the **pAPI** system will provide solutions for how to deliver this service.

The choices made to solve the architectural limitations will allow pAPI to integrate technologies and methodologies that are modern and utilise the built-in features of the hosting environments for security and deployment.

There are many structures within the **pAPI** system that will be developed to deal with the challenge of supplying safe, secure, and useful APIs to the users of this application. The initial step is to identify the two main process that occur within **pAPI**.

Firstly, the **pAPI** system will call APIs from API developers and store the JSON data in a NoSQL database (MongoDB). The second process will be to manage the data distribution from the database to the UI (REACT). This document will outline the infrastructure that is required to deliver on these concepts. While considering this, the developers will also have to adhere to ISO and IEEE standards and recommendations regarding security, maintainability, and reliability.

pAPI's long-range vision is that all users will be able to access valuable APIs from developers in one location without the hassle of registering for membership, logging in, or securing individual API keys to conduct calls themselves. **pAPI** will provide easy to navigate API filter option that will improve the customer experience. **pAPI** will provide for the user a comprehensive catalogue of APIs in a comfortable format that provide the user with the most relevant API for their needs.

The **pAPI** overall goals are as follows.

Project Goals Include:

1. A search feature, comparable to a google search aimed specifically at **pAPI's** library of APIs.
2. A filter system that can contains categories, tags, or key words.
3. A display that is eye catching and easy to navigate
4. A comprehensive format that allows for full discovery of a specific API.
5. A database that can cache an API from the developer in its entirety.
6. A process that will call APIs to update regularly in the database
7. A user guide that methodically provides best practice for the use and implementation of APIs.
8. A link that is direct to the API developer if the user would like to secure their own key.
9. The ability to use snippets immediately

The **pAPI** system should be designed with three separate modules. This separates responsibility and provides a line of demarcation for the development. These three modules are the Front-end or User/Client side and the Back-end (which has been separated into two parts). The Back-end represents the acquisition of APIs and Managing the database to which they are delivered.

The pAPI Modules

REACT Front End

- Search Bar
- Category Display and formatting
- Thumbnail representation of API and Formatting
- Filters to include
    - Tags
    - Themes
- Formatting so that all useable code can be accessed
- Direct links to API developers
- Easy UI Navigation
- Enhanced UX

pAPI Database (*MongoDB*)

- Organised to receive API Data
- Cached API data

- Collates Endpoints

pAPI system

- Calls APIs with relevant keys
- Collects Endpoints
- Processes pertinent to security
- Process pertinent to reliability
- Processes pertinent to maintainability
- Processes pertinent to functionality

pAPI requires a design with specific functionality to model internal business processes, work and data flows, and integration requirements. This document will clearly outline the elements that are required to develop a robustly designed and effective product.

Audience

The intended Audience for the SDD is the project manager, project team, and the development team.

- Poni Sula: Project Manager
- Kurtis Denning: “React” developer
- Oliver Schweikert: “MongoDB” and server-side functions developer
- Waseem Ahsan: Testing and quality control Manager.

User Types

It is essential to identify all of the parties that will be affected by the designs implemented by **pAPI.** This understanding can provide insight for the developers as they construct the processes and interfaces.

pAPI will be designed to accommodate the needs of many different users, with features and functions appropriate for each one.

- ***Professional Developers*** are individuals in need of specific useable snippets or extended access to full API data. Their intention will be to implement code into their developments. This may be in a short-form (snippet) or in a long-form (Wider or complete API use).
- ***“Interested Party”*** users are individuals such as online gamers who want to draw down their personal statistics or metrics about their game play.

They will know what they want but may not understand how to implement the API's or what part they require.

- ***Novice users*** are individuals are those who have some technology awareness but want to browse or research the options and opportunities afforded by APIs.
- ***Explorers*** are individuals who may have limited experience with technology and want to investigate or research what APIs are and how they work.
- ***Administrators*** are individuals or groups that provide support for the maintenance , updating and administration of the application.
- ***API developers*** are individuals or groups that generate the APIs, they will be able to view their APIs and see how they are represented.

APIs are supported in pAPI

The premise of the **pAPI** application is to source and store APIs. It is then a requirement of the **pAPI** application to be designed around the use of APIs. APIs will be supported within and as the theme of the application.

Design Constraints

Given the nature of the project there are some critical and non-critical design constraints that must be considered. This SDD highlights and addresses the known constraints facing the project team.

Performance, storage, security, and access are all elements for consideration, and they represent limitations on the ability of the design team, some of these constraints are addressed here.

Financial

The most prominent constraint will be the financial constraint. As the application gains more users the ability to scale the product to cloud servers will be required. This will come as a financial burden to the project team. Several strategies will be employed to mitigate the cost of hosting this application on the cloud. Utilising the different tiers of different service providers will be a key strategy to relieve the financial burden on the development team, and the initial consumers.

Technical

Specific skills and understanding of mobility, "demand response" management, optimization and server management are required by the team as a whole and more specifically the development team tasked with coding and structuring the solutions.

By controlling the process and allowing time for research **pAPI** project management has diminished the impact of some of the technical constraints.

Due to the limited scale of the application and the light resource requirements at the outset. We do not foresee any hardware, network, internet, or database maintenance challenges.

External organisations

The proposed application focusses on public APIs and multiple API developer websites, this will require a degree of coordination and collaboration with those parties. Monitoring API call volumes, rights, and licensing compliance, and up-to-date data has been identified as points to consider.

In order to comply with developers' conditions and obligations **pAPI** will use its keys/tokens on a scheduled update calendar. This will limit the possibility of spamming the API developers' sites with too many requests.

System Architecture Issues

This section provides clarity on the influences of design in regard to architecture. This section will address the dependencies, risks, and goals of the project. This segment will also provide a representation of the theoretical infrastructure that will guide the developers.

How the particulars of the system come together and will be scaffolded are discussed in this section. We acknowledge that there are some considerations that need immediate recognition and others that can be resolved as they occur. The system architecture design plan addresses these issues.

Dependencies

The **pAPI** application can be dependent on third-party systems. These third-party systems are in relation to API developers and the use of APIs. Although the business decision to have a limit on the number of updates in a set period of time may mitigate the use of the developer's APIs there still exists some risk.

The third-parties will be contacted, and agreements reached (where possible and practicable) around the use of their APIs.

Third-party Considerations:

- API developers’ websites
- API developers APIs

Risks

Because **pAPI** will hold the API keys to call the relevant APIs into the database, the most obvious risk is being banned from API calls. Overuse or over calling (Spamming) can be punished by having the key or IP address banned by the developers’ sites. Limiting the updates to a schedule will ease this risk for **pAPI**.

This risk exists not only for the API developers but also for the **pAPI** system as well. To monitor and diminish this risk, each user will be stamped in real time with an id and if abuse or overuse occurs the individuals ip address will be blocked.

There are also risks with the use of hardware versus using the cloud infrastructure, in the free tier of Heroku, and AWS volume is important. As and when engagement with the web application occurs more secure and capable tiers will be employed.

As an important consideration and part of the **pAPI** business model. API data will be regularly updated. However, prior to each time scheduled update the data may not be 100% accurate, relevant, or current.

Risks:

- Overuse (Spamming) API Developers web sites.
- Overuse (Spamming) **pAPI** web site.
- First tier functions from Heroku, AWS.
- Up to data API data

Architectural Design Goals

It is important to have clear and attainable goals for the development team. This will provide project structure and allows for the team to identify clear milestones, and in turn deliverables.

So, to guide the development team with a strong outline the design goals are defined here.

Development environment:
The application environment must remain consistent. This will alleviate any adverse effects of interoperability or quality.
Useability:
The features of **pAPI** must demonstrate ease of use and provide a powerful user experience. The features cannot divert too greatly from the user expectation unless it is an improvement in the existing models. This translates to mean **pAPI** will not disrupt any long-standing conventions in the API market, however if an innovation or solution that we create is superior to an existing standard then we will implement it to improve the customer experience.
Extensibility:
pAPI Features must be extensible. Any features can be enabled if the user requires them.
API Enabled:
The application must be API centric and support an open and published API architecture.
Restful Framework:
The underlying architecture must be a REST framework. The **pAPI** application must also conform to the constraints of REST architectural style and allows for the interaction with RESTful web services.

Operational Environment
This is the suggested release environment, each element has been chosen with regard to its flexibility, and potential scalability.

- Git Version Control
- GitHub Repository
- MongoDB Database
- NGINX Server (as applied by Heroku)
- Apache HTTP Web Server (as applied by Heroku)

Development Environment
As different to the operational environment, the development environment is an opportunity make changes to the code without adversely affecting the released products. We have remained as consistent as possible in this regard

and aligned the operation and development environments for an easy transition.

Software	Description
GitHub	Version Control repository
Heroku	Cloud Computing Platform
MongoDB Atlas	Database
Chakra	UI Framework/Theme
REACTJS	Programming Framework for Web UI
Gin	GO-Lang Framework
Redis	In-Memory Datastore

Table 1 Development Environment

Development Methods

The best practice architecture is to layer the application into multiple autonomous applications that can be replaced individually and allow us to keep the application running while we are paying attention to any specific layer.

Scalability:

Once the traffic increases we should be able to add or remove servers as the solution requires.

Availability:

Infrastructure redundancy is required to support a high availability environment.

Security:

All of the back-end code should be securely stored and not exposed for bad faith actors.

Extensibility:

Each module should have its own set of responsibilities as not to be reliant on a past or future module.

Restful Framework:

Simple flexibility is the target. Being non-reliant on a specific language and architecture. The architecture must be able to account for the best suited language for that layer.

Cloud Strategies

The use of cloud technology has formerly been the most disruptive forces impacting applications and infrastructure. The cloud forced business model and new architecture decisions which have impacted the way **pAPI** can deploy, manage , maintain, and protect their data.

Heroku offers multiple options for provisioning IT infrastructure and the deployment of web-based applications.

AWS provides and redundancy for data storage.

Infrastructure on Demand:

The building blocks that represent the infrastructure are already provisioned as required, they can follow actual demand and allows pay-as-you-go. This means greater flexibility for the development team in all development environments as well as disaster recovery scenarios. The development team can code in solutions that can be automated which will improve self-service and maintain the delivery of desired business and technical outcomes for consistency.

Cloud Computing:

Cloud computing today has brought on-demand network access to a shared pool of configurable computer resources. This can be rapidly provisioned and released with minimal additional effort from management.

For both AWS and Heroku offer global infrastructure on a pay as you go model. The first tier of which is free and as the demand on resources increases so does the cost. This has provided massive flexibility in meeting the requirements for disaster recovery and data protection.

Recovery:

A physical environment poses added risks to secure and safe disaster recovery and general security. Cloud strategies allow for multiple solution to disaster and security scenarios to be generated within the cloud environment. Testing and Development can be conducted on the cloud, this can provide levels of security not afforded to a physical environment.

Mobility:

The way in which we conduct business and in general our lives has been heavily influenced by mobile technology and mobility in general. With cloud considerations combined with Mobile technology more accurate information is available more immediately than ever before. The office has become and where you can carry your phone and get a message. This must be a consideration in the planning of any software engineering project.

Scalability:

Going viral is a term used to indicate the rapid engagement of an idea. As will applications that can have instant success or a more graduated evolution the data management solution must adapt to the requirements it is tested by. The system must generate business value that is in accordance with quick and efficient changes that may be required depending on the applications performance.

Archive:

The cloud provides a secure location to support either archiving valuable information in a physical location or as a dynamic multiple level solution on the cloud.

PaaS (Platform as a Service):

Heroku and AWS have built in solutions, this includes software development tools, application servers, analytics, network connectivity, database management and much more.

Systems Descriptions

The development, and operational environments and strategies around cloud usage and application lend themselves to the description of how the system will be employed. This will influence what specifications these environments will require. The following table briefly describes the systems being employed.

	System	***Note***
Programming Language	*Go Lang*	*Go-lang is an open-source language that is supported by Google. It is easy to learn and get started. It has built in concurrency and a robust standard library. It has a growing ecosystem of partners.*
Programming Language Framework	*GIN*	*Gin is a web Framework written in Go-Lang. Radix tree-based routing with predictable API performance. Crash-Free, with JSON Validation and error management.*
Development & Version Control	*Git*	*Git is a free and open-source distributed version control system designed to handle everything from small to large projects with speed and efficiency.*

Hosting Service	*GitHub*	*GitHub is a code hosting platform for version control and collaboration. It allows you to work with others from anywhere in the world. Projects can be hosted directly from your repository.*
Database	*MongoDB*	*MongoDB is built to scale out architecture that has become popular with developers of all kinds. It is document based and allows developers to store unstructured data.*
Datastore	*Redis*	*Redis is an open source in memory data structure store, used as a database cache and message broker.*
Web Server	*NGINX*	*NGINX is a web server that can also be used for multiple solutions. It is free and open sourced.*
Web Server Software	*Apache*	*Apache is free and opensource software it is one of the most popular web server software's on the planet.*
Cloud Application Platform	*Heroku*	*Heroku is a platform as a service first that enables developers to build, run and operate applications entirely in the cloud.*

Table 2 System Description

Server Requirements

Server power, storage capacity and flexibility in a changing commercial environment are issues that should be addressed before the development team begin. Planning for expansion can only take place if the foundation of the system structure is sound.

As usage and engagement in the initial stages may be low, a practical server is required. With Heroku this means that the "free" tiers will be used until the traffic to the application can justify an increase in dedicated CPU power, memory, and storage. Horizontal scalability can be achieved by stepping into the next "hobby" level, this can also be scaled up with the remaining paid hierarchy within Heroku.

AWS also offers similar levels of introduction and scaling. This is all dependent on usage. So, after the initial phase of introduction, it will become user-pays with both AWS and Heroku. A cost consideration dependent on engagement and usage.

With regard to AWS and Heroku their multi-tiered user pays practices are perfect for a new application. The following table represents the server specifications for the **pAPI** application.

Device	Web Server	Storage	OTP Server	Database Server
Type	Heroku	AWS S3	Heroku	Heroku
MEMORY	4GB	5GB	4GB	4GB
STORAGE	68GB	100GB	68GB	68GB

OS	Ubuntu(LTS) v20.10	Ubuntu(LTS) v20.10	Ubuntu(LTS) v20.10	Ubuntu(LTS) v20.10
SOFTWARE	Apache		pAPI	MongoDB

*The estimates have been taken from the Heroku Performance Characteristics for standard packages. https://devcenter.heroku.com/
**The estimates have come from the basic package offered by AWS. https://aws.amazon.com/

Table 3 Server Requirements

Decomposition Description

To provide some context for the developers in their construction of the **pAPI** application some diagrams have been generated. These represent operational scenarios

pAPI Use Case UML

Understanding how the various users will utilize the **pAPI** application will provide the developers with insight into the UI and UX designs. The Use case below demonstrates multiple users and how the structural processes and elements interact. The use model demonstrates that there are three distinct Actors, the User, the **pAPI** Admin, and the API developers. These actors work with the **pAPI** system and within that wider system there exists two more structures that interact harmoniously with each other. The REACT front-end React interacts directly with the user and provides features such as browse, search, feedback, and request. The React front-end simultaneously co-operates with the back-end formatting the relevant data for display. In this instance the user will not directly be interact with the back-end functions. The Back-end maintains the database, calls updates for APIs, and manages the server side functions. The API developers interact only with the issuance and calls for APIs.

As a whole the Figure 1 “UML Use Case General” can demonstrate many user stories and how process structures can exist within a wider system. Other users’ stories and more in depth use cases have been discussed in other documents and for the purposes of this SDD it is provided as reference.

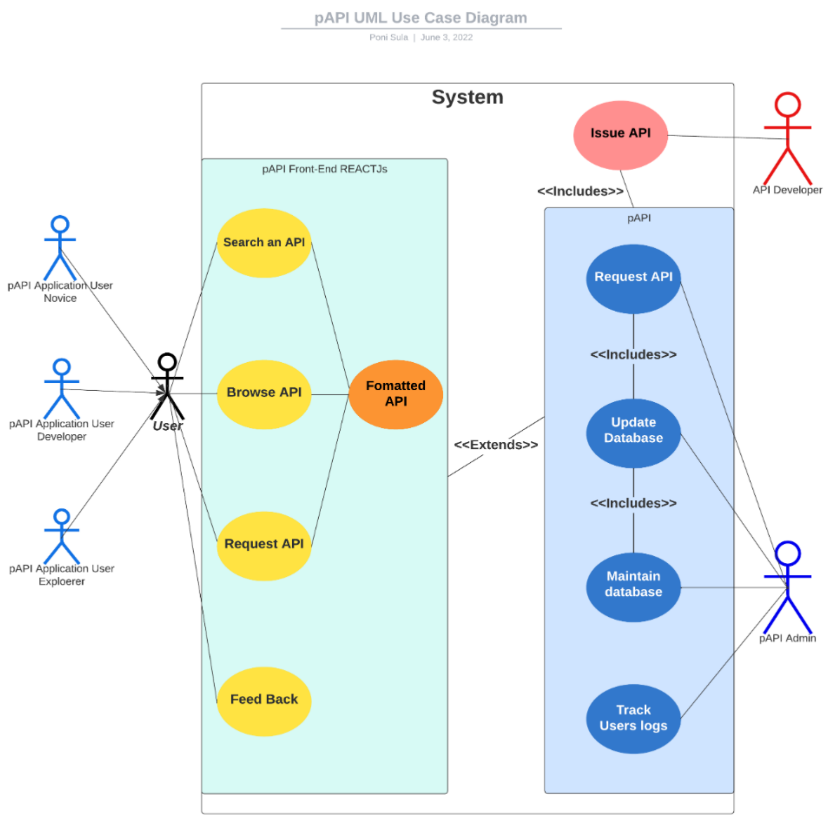


Figure 3 pAPI UML Use Case General

Use case expansion.

The React Module allows web users to review library contents quickly and easily.

There are two scenarios:

- Specific search
- Discovery (Browse)

Common Actions

- Search
    - Easy and flexible search functions
    - Once customer retrieves requested API they will
        - Review Data
        - Use Snippet
        - Visit Developers URL
- Browse
    - Easy and flexible search functions
    - Once customer retrieves requested API they will
        - Review Data

- Use Snippet
- Visit Developers URL

Theme	I want to…	So That….	Use Case	Notes
User Interface	Access the pAPI REACT Model.	I can review the library of APIs	The landing page appears with search, browse and examples	UI Design will be critical for module accessibility.
User Interface	Search for API	Review an existing API	A user with knowledge of what they are searching for is provided with the specific item they searched or a not available notification.	Search UI must be simple and fast. Advanced search criteria required.
User Interface	Browse Display	Review Library Contents	A user can view the catalogue of APIs	UI design will be critical so that the review of the library is engaging, informative, and accessible.
Function	Update Database	The database is Current and relevant.	The pAPI system maintains relevance and accuracy	Server-side functions to update the database for display regularly.
Function	Call API for Update	pAPI calls for the latest API data, and updates database.	pAPI ensures reliability by caching API data	Server-side function assures accessibility and reliability.
General Data	View formatted API data	Calls from React to pAPI are accurate and display relevant information	Front end PAPI is accurate and relevant	UX design so that the user is unaware of the back-end processes to maintain display and search functions.

Table 4 User Models

Hardware Architecture

As discussed in a previous section, hardware requirements are minimal as the cloud servers provide more flexibility and accessibility. Different deployments can run on different server configurations , at this stage **pAPI** will typically be deployed on three servers.

- A web server running Apache to host the web application. In this instance we are considering GitHub has this capability.
- A storage server for application configuration files.
- A site-specific server for **pAPI** and its database.

Program Structure Design

The overall architectural plan provide insight into three key areas. System and security framework, performance, and software architecture.

The following diagram (Figure 2 “High Level View of the Architectural Plan”) outlines the basic modules and the processes each are responsible for. AS can be seen in the figure, the base or simple model has a user accessing the **pAPI** system unaware of processes that maintain the **pAPI** library.

The second image in the figure further breaks this down to identify the specific modules that will be created and developed. The User via Mobile app can only browse the APIs and the user using a laptop or PC can access the APIs and its useable data (Snippets).

The **pAPI** system is separated into a Front end which is powered by GitHub and designed in React JS. This displays the UI and provides the UX. The database calls a made through the "Buttons" developed in the interface and can access all the API information held within the **pAPI** NoSQL database.

The **pAPI** system them separates its two main processes of call APIs and Delivery them to its Database for management and storage.

This simple overview gives the developers clear modules, and processes to understand.

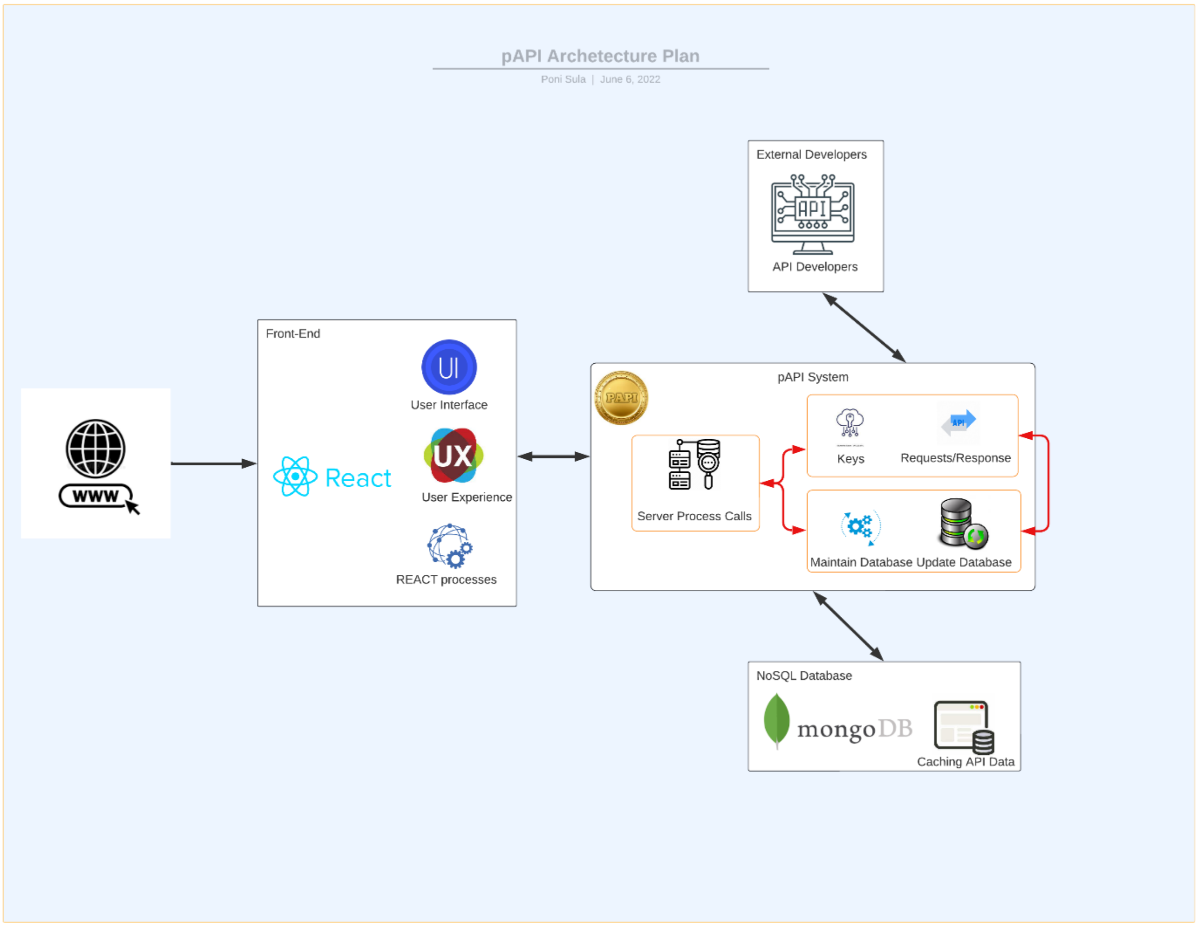
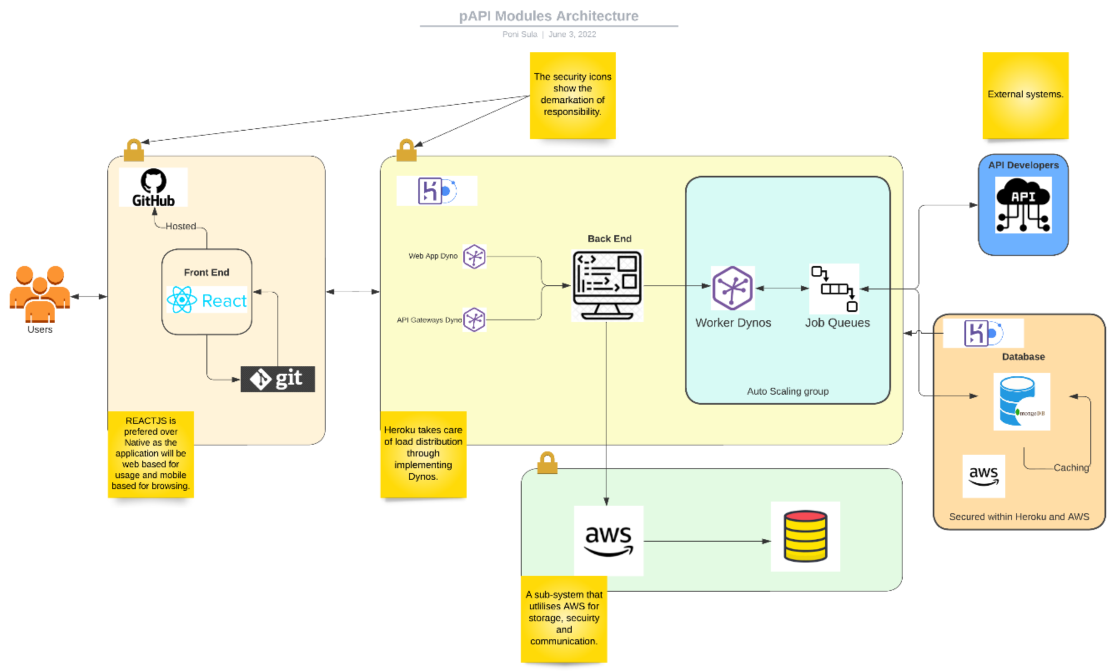


Figure 4 High Level View of Architectural Plan

System and Security

Figure 3 “System & Security High level View” is a more detailed look at the system provided by **pAPI**, with specific reference to the environments it is developed and maintained within. Built into GitHub, Heroku, and AWS are security, storage, and analytics divisions. This allows each module of **pAPI** to be self-contained.

Features that are built into GitHub, Heroku, and AWS add security to the **pAPI** application. This does not preclude more features being added to the **pAPI** code. It provides multiple levels of reliability, security, and maintainability.



pAPI Modules Architecture
Poni Sula | June 3, 2022
The security icons show the demarkation of responsibility.
External systems.
Users
GitHub
Hosted
Front End
React
git
REACTJS is prefered over Native as the application will be web based for usage and mobile based for browsing.
Web App Dyno
API Gateways Dyno
Back End
Worker Dynos
Job Queues
Auto Scaling group
Heroku takes care of load distribution through implementing Dynos.
API Developers
API
Database
mongoDB
aws
Caching
Secured within Heroku and AWS
aws
A sub-system that utilises AWS for storage, security and communication.

Figure 5 System & Security High Level View

Performance

Heroku and AWS S3 are utilised for their hardware performance and reliability. S3 is storage for the internet and Heroku is services storage for functions from web applications. Both are highly scalable, dependable, and low latency data storage infrastructure with a free tier.

S3 is highly flexible, it can store any amount of data you want and can be used for emergency disaster recover.

Heroku is a polyglot platform and operates as PaaS.

Provided in synergistic partnership performance for **pAPI** is world leading and innovative.

Software

As stated earlier in this document regarding separation of responsibility and the establishment of modules. The Software architecture can be described in terms of Tiers. **pAPI** is a user facing application, it will be constructed using three tiers.

The tiers will comprise of the presentation tier (Module 1) the data tier (Module 2) and the logic Tier (Module 3).

The presentation tier represents components that users directly interact with.

The Logic tier contains the code required to translates processes and actions at the presentation tier.

The data tier consists of storage media and holds the relevant data to the application. (Databases etc.)

Security Software

There are a number of principles applied to the proposed system security.

- Apply security to all layers:
  - Rather than running firewalls only at the edge of the infrastructure, firewalls and other security controls are on all resources.
- Enable traceability
  - All actions and changes to the environment are logged and audited.
- Implement a principle of least privilege.
  - Ensures that authorisation is appropriate for each and every interaction with the AWS and Heroku resources.

Performance Architecture

Providing performance guidelines will allow the developers to build testing, provide framework for integration, and change management. Scalability is planning for future performance, while performance of the existing system needs to be controlled. Assessed and adjusted throughout the development process.

Desired Quality	The system should predictably execute within its performance profile and manage increased processing volumes if required.
Concerns	• Scalability • Predictability • Response time • Throughput • Peak Load behaviour
Activities	• Conduct Practical testing • Assess against requirements • Create performance models • Rework Architecture if required
Strategies	• Prioritise processing • Distribute processing • Use Asynchronous processing • Partition • Minimise shared resources • Optimize processing • Make compromises
Pitfalls	• Unrealistic models • Inappropriate portioning • Concurrency • Transaction overhead • Imprecise Performance

Table 5 Performance Architecture

System Behaviour

How **pAPI** operates or behaves is crucial for the development team to understand . The following diagrams offers some insight into the anticipated data management, and information flow.

The following sequence, Activity and State diagrams provide context for the processes and development modules for the development team.

Sequence Diagram

Referring to Figure 4 “**pAPI** Sequence Diagram” depicts the simple interactions between the designated objects in a sequential order. The **pAPI** system expresses, after the initial activation by the user, that there are two processes running concurrently to keep the database up to date, and to provide filtered and formatted results to the User Interface.

In Explanation, the Fig.4 shows that the web browsers interactions are contained to actions performed by the user (Search’s and Category link activations), these are processed in the **pAPI** system and returned to the front end or web browser for display to the user.

Simultaneously the **pAPI** system also communicates with the API developers maintaining up to date data. The **pAPI** system activates in two ways, firstly as a response to a query from the user, and secondly on a scheduled update. Both cause the **pAPI** system to activate calls to the API Developer for the most current version of the API data. This acts both independently and in unison with the user requests

The swim lanes show the activations, messaging, and returns in the two processes.

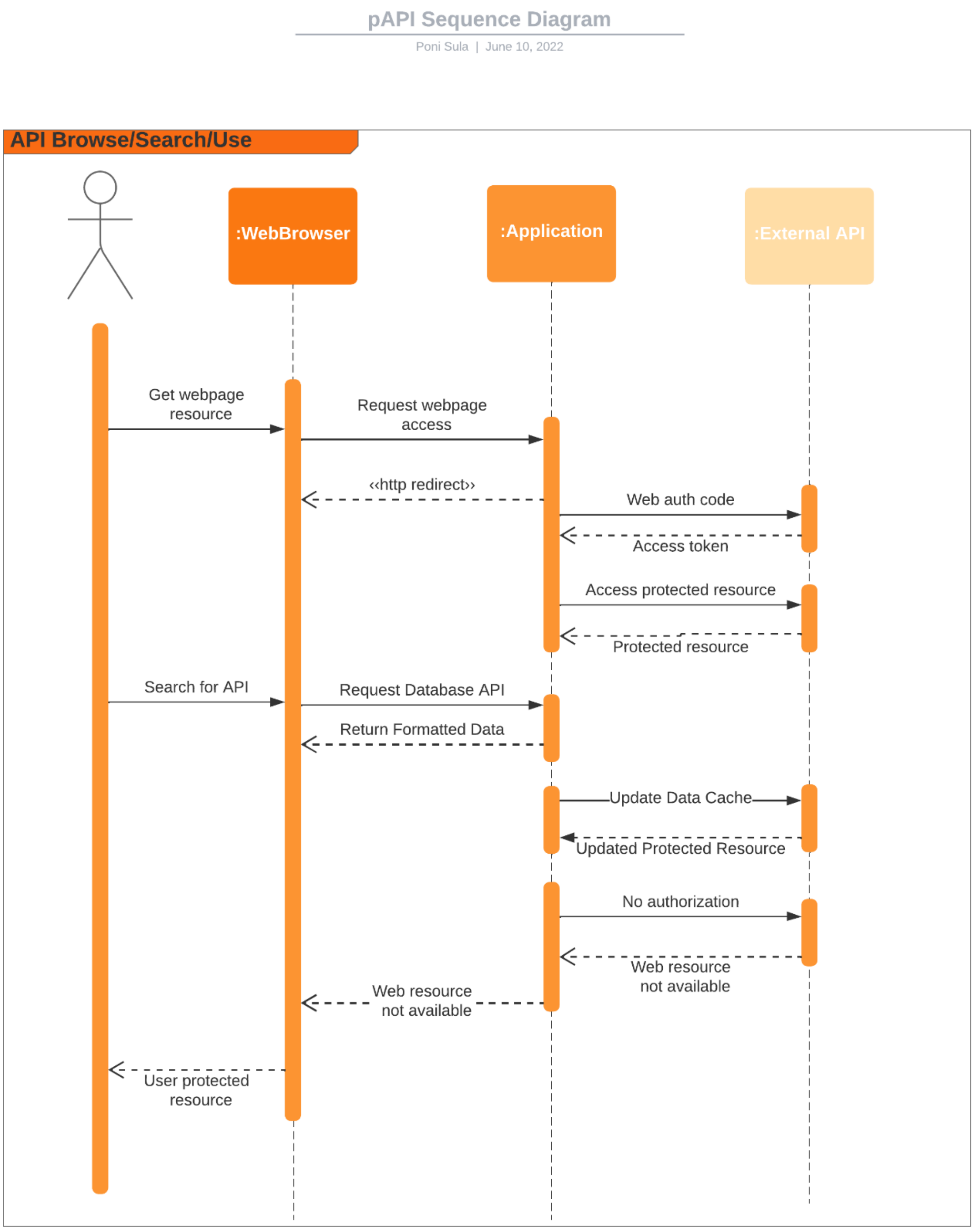


Figure 6 pAPI Sequence Diagram

State Diagram

In reference to Figure 5 “**pAPI** State Diagram” demonstrates the behaviour of the **pAPI** system as responses to events. The simple model of the **pAPI** system shows the major actions are User Request and Data update, these generate responses that change the state of the system. Once the server is activated all input by the user is a request, and all following actions are responses via updates or display.

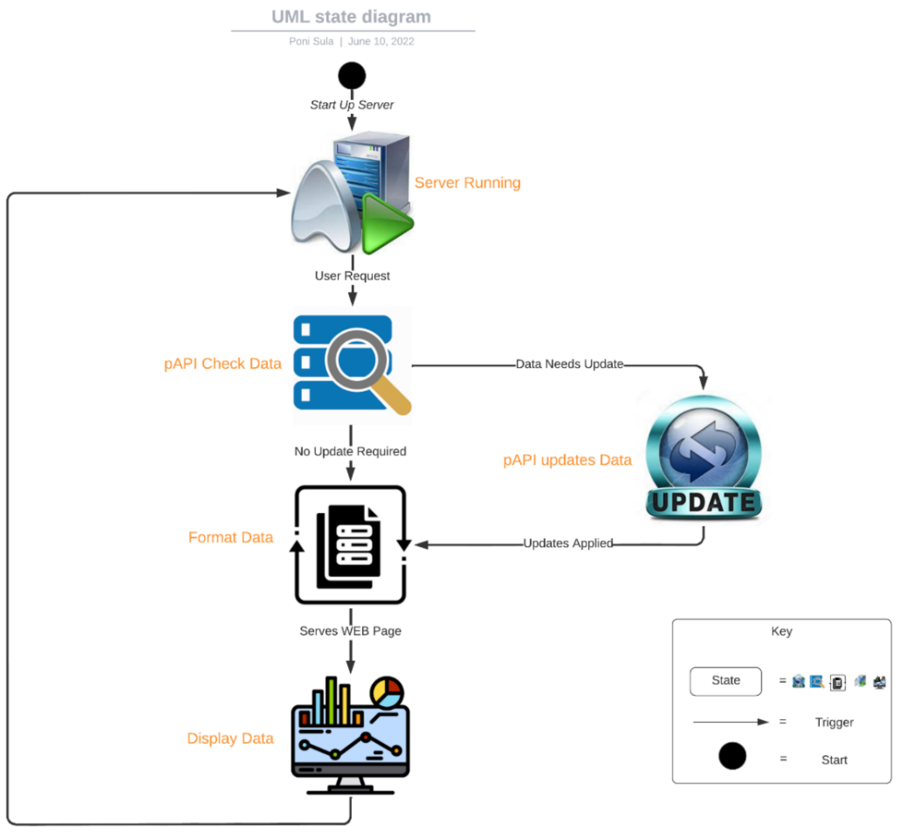


Figure 7 pAPI State Diagram

Activity Diagram

In reference to Figure 6 "**pAPI** Activity Diagram" describes the flow of control in the **pAPI** system. This is another graphical depiction of the processes and reiterates the base messaging and responses within the **pAPI** system.

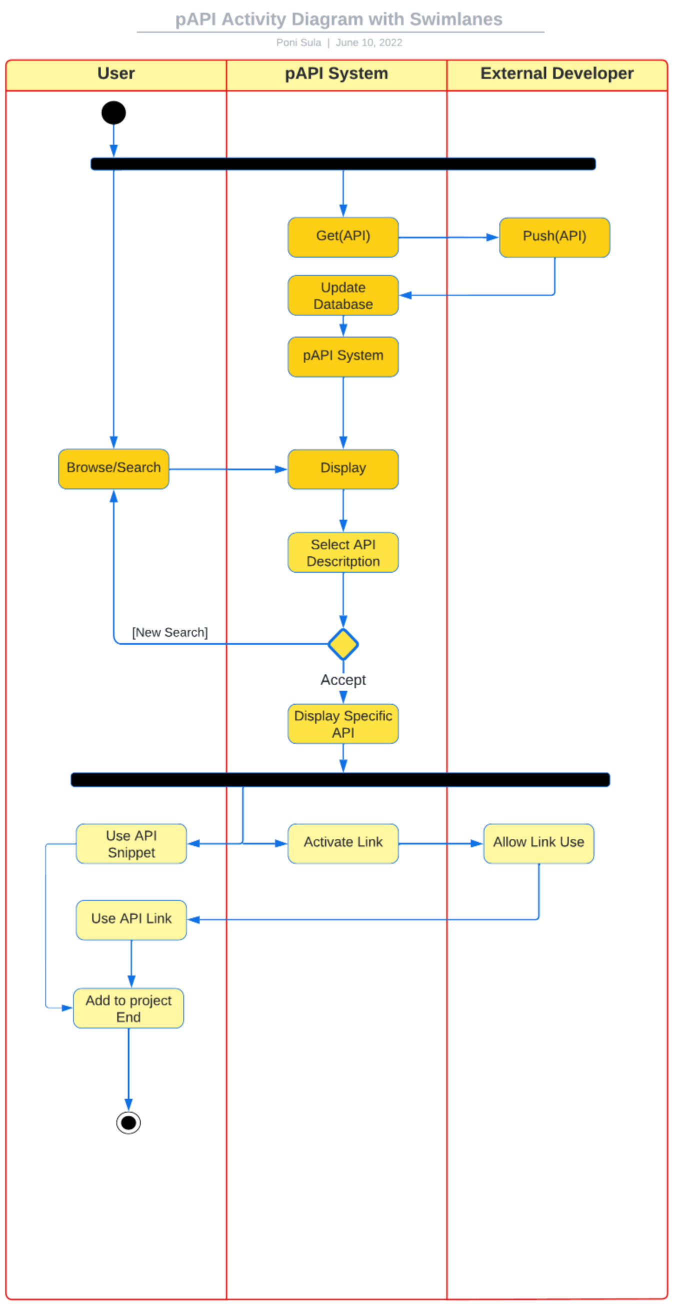


Figure 8 pAPI Activity Diagram

Data Design

Data design is the first step in the design implementation. This will result in less complicated, modular, and useful program structure. The data classes, objects, attributes, and relationships represented in the entity relationship diagrams and the information in the data dictionary are the basis for the data design.

The structure of the data can be viewed at three levels, program component level, application level, and the business level.

The component level: The data structures and algorithms required to manipulate them is necessary.

The application level: This is critical to convert the data model into the database so that the business objectives can be achieved.

The Business level: The collection of information stored in the different databases should be recognised into data warehouse, which enables the data drilling for the business.

pAPI is using dual databases, MongoDB for API storage, and PostgreSQL by default for data security, maintenance, logging, and incidental data storage.

Information Architecture

The following diagrams and chart (Class Diagram, Entity Relationship Diagram, Data dictionary) have been presented in a traditional form so that the development team can understand the relationships and the class structures. Mongo is a non-relational NoSQL database technology; relationships are not applied by the engine itself. The data itself contains relationships, and The ER diagram provides a visualization of this relationship.

Internal data structure

Mongo makes use of collections and documents. The documents will consist of key value pairs which are the basic unit of data in Mongo. Mongo DB makes it easy to store structured and unstructured data. It uses JSON format to store documents.

Global data structure

By utilising a Mongo, JSON is the currency of the application. Both the front end and the backend can deal with JSON data and as it is stored in JSON like structure in Mongo only formatting is required for delivery and display.

Temporary Data Structure
All temporary or analytical data is stored in a more traditional RDMS as supplied by the Heroku, and AWS tier systems.

Class Diagram:
The Class diagram presents nine classes with their attributes and suggested functions. Each class represents an object that is required. User, Language, and category are durable objects that require no functions. The remaining classes have specific functions that aid in the data mining, and operation of the database and the **pAPI** system.

ER Diagram:
The ER Diagram has combined some of the classes for ease of use within MongoDB. The diagram demonstrates that two of the stable Objects, RemoteAPI and Category , have been minimised and transformed into JSON Arrays. This facilitated a more cohesive and manageable document database.

Data Description

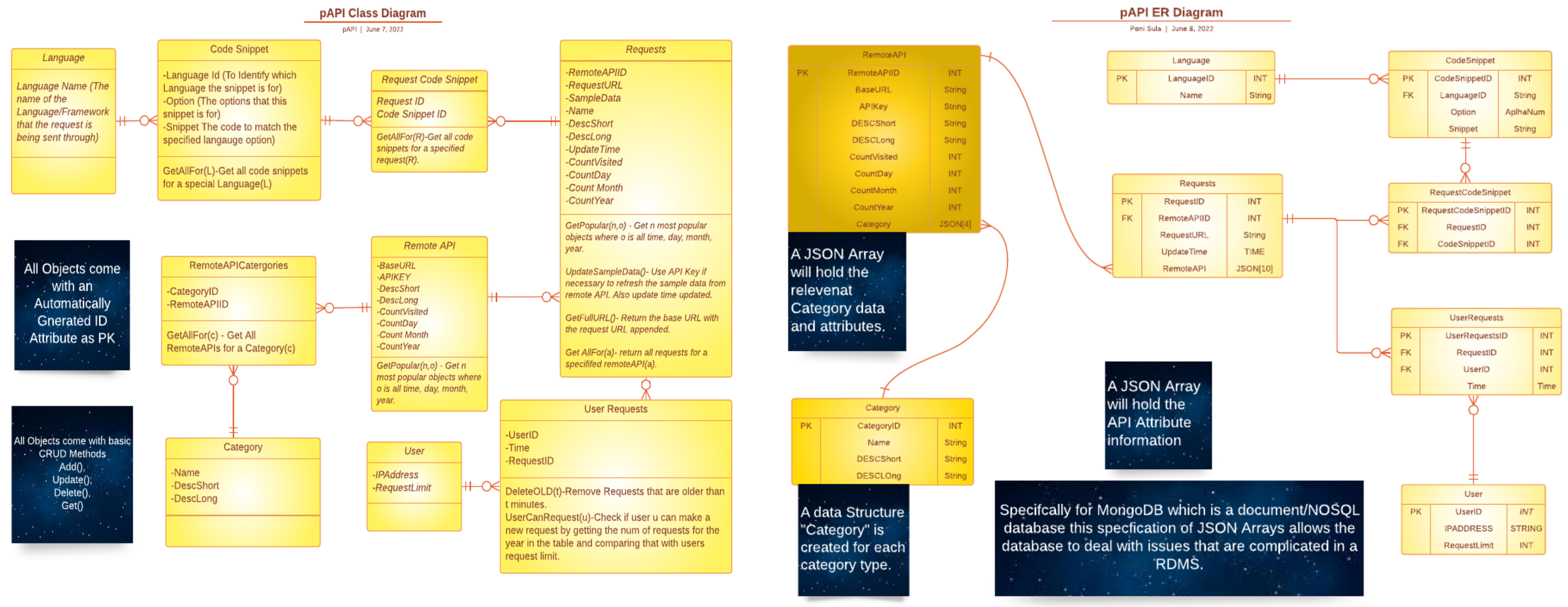

pAPI Class Diagram
pAPI | June 7, 2022
Language
Language Name (The name of the Language/Framework that the request is being sent through)
Code Snippet
-Language Id (To Identify which Language the snippet is for)
-Option (The options that this snippet is for)
-Snippet The code to match the specified langauge option)
GetAllFor(L)-Get all code snippets for a special Language(L)
Request Code Snippet
Request ID
Code Snippet ID
GetAllFor(R)-Get all code snippets for a specified request(R).
Requests
-RemoteAPIID
-RequestURL
-SampleData
-Name
-DescShort
-DescLong
-UpdateTime
-CountVisited
-CountDay
-Count Month
-CountYear
GetPopular(n,o) - Get n most popular objects where o is all time, day, month, year.
UpdateSampleData()- Use API Key if necessary to refresh the sample data from remote API. Also update time updated.
GetFullURL()- Return the base URL with the request URL appended.
Get AllFor(a)- return all requests for a specifed remoteAPI(a).
All Objects come with an Automatically Gnerated ID Attribute as PK
RemoteAPICatergories
-CategoryID
-RemoteAPIID
GetAllFor(c) - Get All RemoteAPIs for a Category(c)
Remote API
-BaseURL
-APIKEY
-DescShort
-DescLong
-CountVisited
-CountDay
-Count Month
-CountYear
GetPopular(n,o) - Get n most popular objects where o is all time, day, month, year.
All Objects come with basic CRUD Methods Add(), Update(), Delete(), Get()
Category
-Name
-DescShort
-DescLong
User
-IPAddress
-RequestLimit
User Requests
-UserID
-Time
-RequestID
DeleteOLD(t)-Remove Requests that are older than t minutes.
UserCanRequest(u)-Check if user u can make a new request by getting the num of requests for the year in the table and comparing that with users request limit.
pAPI ER Diagram
Poni Sula | June 8, 2022
RemoteAPI
PK RemoteAPIID INT
BaseURL String
APIKey String
DESCShort String
DESCLong String
CountVisited INT
CountDay INT
CountMonth INT
CountYear INT
Category JSON[4]
A JSON Array will hold the relevenat Category data and attributes.
Category
PK CategoryID INT
Name String
DESCShort String
DESCLOng String
A data Structure "Category" is created for each category type.
Language
PK LanguageID INT
Name String
CodeSnippet
PK CodeSnippetID INT
FK LanguageID String
Option AplhaNum
Snippet String
Requests
PK RequestID INT
FK RemoteAPIID INT
RequestURL String
UpdateTime TIME
RemoteAPI JSON[10]
RequestCodeSnippet
PK RequestCodeSnippetID INT
FK RequestID INT
FK CodeSnippetID INT
UserRequests
PK UserRequestsID INT
FK RequestID INT
FK UserID INT
Time Time
A JSON Array will hold the API Attribute information
Specifcally for MongoDB which is a document/NOSQL database this specfication of JSON Arrays allows the database to deal with issues that are complicated in a RDMS.
User
PK UserID INT
IPADDRESS STRING
RequestLimit INT

Figure 9 Class Diagram & ER Diagram

Data Dictionary

Field Name	Data Type	Data Format	Size	Description	Example
LanguageID	INT	XX-XXXX	6	A Unique Code to represent the language being used	01-1234
Name	String	XX…..	25	The Commonly known name for the Language	Go-Lang
CodeSnippetID	INT	XX.XX.XX	6	A Unique Code that identifies the Specific Snippet from other Data	01.00.00
Option	AlphaNum	XXXXxx	7	The options that are available for this snippet	A1
Snippet	String	XXXX…	150	Code that is specifically matched to the API	
RCS-ID	INT	XX-XXXX	6	A Unique Code to represent the Request Code Snippet	10-0000
RequestID	INT	XX-XXXX	6	A Unique Code to represent the Request ID	20-0000
RequestURL	String	xxx.xxx.xxx	35	The URL that represents the API Data	https://official-joke-api.appspot/random_joke
UpdateTime	Time	XX:XX.XX	8(Max)	The Time stamp provided for dating updates	12:34.45
RemoteAPI	JSON[10]	[X,X,X,X,X,X,X,X,X,X,X]	JSON Array	This holds the Attributes of the Remote API, this may Vary.	[ID, BaseURL, APIKey, DescriptionSHRT, DescriptionLong, CountVisited, Day, Month, Year, Category(JSON[])]
Time	Time	XX:XX.XX	8(Max)	Timestamp For Request	4:29.00
UserID	INT	XXXXXX	6	Unique ID in Numeric Form	123456
IPADDRESS	String	XX.XXX.XXX	11	The Internet Portal Address	16.903.066
RequestLimit	INT	XXXXXX	12	The Nominated number to represent the ceiling limit of requests.	1000
APIID	INT	XX-XXXX-XX	10	A Unique Identifying Number as Identfication	99-1234-00
BaseURL	String	XXX.XXX.XXX//	25	A URL that acts as the base URL for the API Delivery	https://official-joke-api.appspot/random_joke
APIKey	String	XXXX-XXXX-XXXX-XXXX	16	A unique Key that allows access to call for APIs from developers	123Y-TYD3-1234-trR4
DESCSHRT	String	XXXX…		A brief Description of the API	This is an API
DESCLNG	String	XXXX…		The Full Description of the API	This is a full description of an API
CountVISITED	INT	XXXX	1000	The Count associated with site visits	21
CountDay	INT	XX	365	The Number of days	8
CountMonth	INT	XX	12	The Number representing the month 1-12	6
CountYear	INT	XXXX	9999	The number representing the Year	2022
Category	JSON[4]	[X,X,X]	JSON Array	This will hold the attributes of a Category. Size should be consistent	[ID, Name, Description Short, Description Long]
CategoryID	INT	XX-XXXX		A Unique Identifier for the Category	77-0987
CatName	String	XXXX…	25	The name representing the category	This is the name of the Category
DESCSHRT	String	XXXX…	150	A brief Description of the Category	This is a brief Description of the Category
DESCLNG	String	XXXX…	String.MaxValue	A Full description of the Category	This is the full description of the Category

Table 6 Data Dictionary

Data Structure Code for Mongo DB

This structure is written in Go-Lang and generates the structs and processes as well. A link to the GitHub repository is provided for further information. https://github.com/oliverschweikert/pAPI. All of the entity creation and modelling has been supplied for review in the GitHub repository.

Human Interface Design

The UI or user interface is the first thing that users see and interact with. With all UI the landing page should be subtle yet carry all the relevant and vital information to engage the user. The interface should convey all the information required to successfully navigate the page and utilise it for the intended purpose.

pAPI is a single page that has a search feature, is browsable, and displays the APIs that are available. A second page will publish when a specific API is chosen from the filtered list. This saves on load times and requires less overhead in resources to publish.

The aesthetic is use of colour and white space and to inform the user of the relaxed and easy-going nature of the site.

Screen Images

Given the philosophy of the **pAPI** team, this scope was narrowed down to a more simplistic appearance, and functional design. Keeping in line with the philosophy of simplicity bridging the gap between form and function. Figure 8 "Figma wire Frame & Design" is an artist's representation of RapidAPI and API List inspired designs.

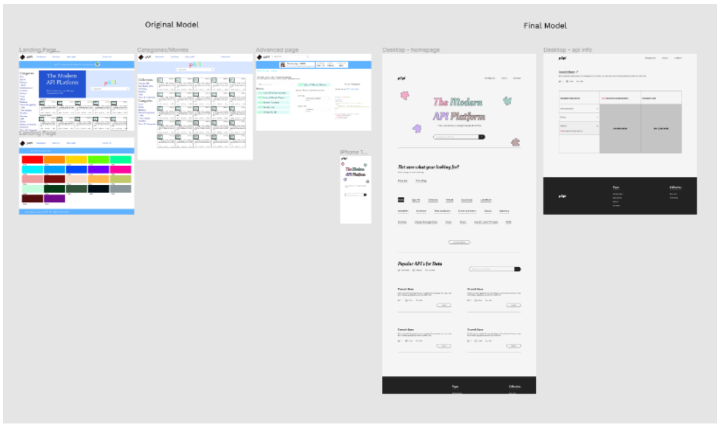


Figure 10 Figma Wire Frames & Design

The designs starting from the left are based on the templates that RapidAPI and API List offered. The designs on the right are the version created by client consultation, and requirements analysis.

Screen Objects and Actions

In place of a site-map we have constructed a developers guide to the form and functions of the key elements on the two pages that will be required to deliver the **pAPI** product. Figure 9 “Site-map Brief” highlights the outline of the two display pages. With sticky notes pointing to the process of the buttons, links, boxes, and frames.

- Nav Bar
  - Basic links to pages
- Footer
  - Basic communication data
- Search Box
  - A functioning predictive, and comprehensive string manager
- Category Links
  - Links to filtered results
- Search Button “GO”
  - Activate search feature
- Display
  - Filtered Results
- Detailed Display
  - The Detailed display Window

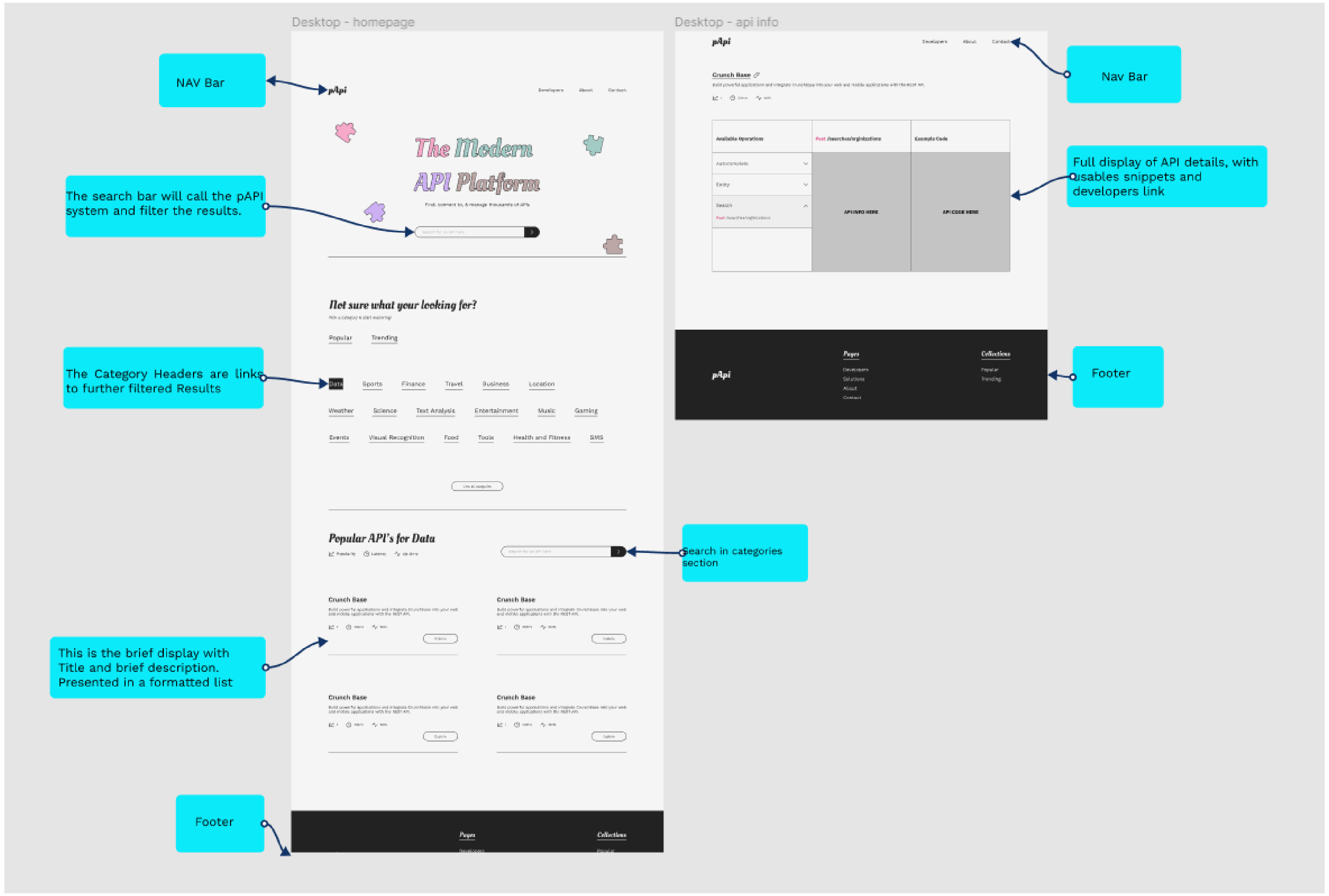


Figure 11 Site Map Brief

Testing Issues

Testing Requirements:

Description

pAPI testing will aid in the quality assurance and functionality of the web application. To ensure that we deliver a commercially viable product, pAPI will provide robust testing of all of the functional features of the software development. Regular scheduled audits will maintain a faults log that will be addressed by the development team on a weekly meeting.

Testing Approach

- Unit testing
- Progress testing
- Version testing
- UI/UX testing
- Last version test

Testing Tasks to be performed

The principal features to be tested are:

- Accessibility
- External Interfaces
- Functional server tasks
    - Acquiring API from developers
    - Configuration of API data.
    - Data structures
    - Data storage
    - Data usage
    - Database construction and utilisation.
- UI engagement and useability
- Assessment of user experience via survey response.

Functional testing

There are three key features of the pAPI landing page. The search Bar, The category Headers, and the brief display of API data. The choice to deploy this as a single page shortens the load time, and navigation time. Table 7 “Functional Testing Case” lists the key features required by the stakeholders.

ID	Input	Expected Results	Actual Results	Status	Comment
FT-1	Type Search String	Display Filtered List	Display filtered List	Complete	Worked as designed
FT-2	Click Category Link	Display Filtered List	Displayed filtered list	Complete	Worked as designed
FT-3	Type Search String in Category Search Bar	Display Short Version Filtered List	Displayed Filtered List	Complete	Worked as designed
FT-4	Click Specific API Link	Display Full Details of API	Displays API information	Complete	Worked as designed

Table 7 Functional Testing Case

Unit/Performance Testing

Using Swagger with Go-Lang will allow unit and performance testing to occur at the same time. Use https://papi-project.herokuapp.com/ to access swagger directly for testing.

ID	Input	Code	Expected Results	Actual Results	Status	Comment
UPT-1	GET All API Data	200	Successful	As Expected	Done	Swagger was an invaluable tool
UPT-2	GET All API Data (2 tries inside 10 sec)	429	Too Many requests	As Expected	Done	Swagger was an invaluable tool
UPT-3	GET Single API Data	200	Successful	As Expected	Done	Swagger was an invaluable tool
UPT-4	GET Single API Data	400	Problem With User request	As Expected	Done	Swagger was an invaluable tool
UPT-5	GET Single API Data	429	Too Many requests	As Expected	Done	Swagger was an invaluable tool
UPT-6	GET Single API Data	500	Problem With Server Processing	As Expected	Done	Swagger was an invaluable tool
UPT-7	GET Single API Data	502	Problem Connecting to external API	As Expected	Done	Swagger was an invaluable tool
UPT-8	GET List categories in DB	200	Successful	As Expected	Done	Swagger was an invaluable tool
UPT-9	GET List categories in DB	429	Too Many requests	As Expected	Done	Swagger was an invaluable tool
UPT-10	GET Single Category in DB	200	Successful	As Expected	Done	Swagger was an invaluable tool
UPT-11	GET Single Category in DB	400	Problem With User request	As Expected	Done	Swagger was an invaluable tool
UPT-12	GET Single Category in DB	429	Too Many requests	As Expected	Done	Swagger was an invaluable tool

Table 8 Unit and Process testing

Integration/System Testing

Integration testing will follow tests that are performed for Functionality.

ID	Input	Expected Results	Actual Results	Status	Comment
IST-1 ref FT-2	Click Category Link	Display Filtered List	As Expected	Complete	As performed in Function test
IST-2 ref FT-4	Click Specific API Link	Display Filtered List	As Expected	Complete	As performed in Function test

Table 9 Integration & System testing

Security Testing

To secure the pAPI system from user input and from unwanted database interference the development team will implement a 10 sec time out for the user and develop an admin Panel for the purpose of securing the Database and the back-end applications.

ID	Input	Expected Results	Actual Results	Status	Comment
ST-1	Access Database Admin	Password Request	As Expected	Complete	The admin portal is for function only.
ST-2	Enter Password	Access to Database CRUD actions for APIs	As Expected	Complete	The admin portal is for function only.
ST-3	Access Database Categories	Password Request	As Expected	Complete	The admin portal is for function only.
ST-4	Enter Password	Access to Database CRUD actions for Categories	As Expected	Complete	The admin portal is for function only.
ST-5	Enter multiple calls on a category link within ten secs	Time-out	As Expected	Complete	This is tested in multiple areas

Table 10 Security Testing

Maintenance and Support:

Performance

- Traffic flow meter
- Request meter
- Format remodelling (Options)

Capability

- Memory capacity for cache
- Server capacity

Scalability

Scalability will be achieved through a move to cloud-based servers.

Technical reviews

Technical reviews will be conducted month on a scheduled basis

Audits

The testing team will conduct audits to assure compatibility and compliance to internal standards

Delivery, Installation, and Acceptance:

Installation

- Users will use the URL and be delivered to a landing page with pAPI display.

Usability

- The search feature will be contained at the top half of the page for easy line of sight and usage.
- All the category tags, type tags, and functionality tags will be incorporated with the search feature and filters.
- The list of the popular APIs will be displayed with the familiarity of YouTube, or Spotify aesthetics and usability.

Acceptance

The acceptance criteria will be a

- 1-3 touch retrieval of specified API.
- Display of minimum four thumbnails with brief synopsis
- One clicks retrieval of listed API.
- Copy & Paste use of Snippet
- Click re-direct to API developer's website.
- Click exit from site

These criteria represent the user acceptance of the product and in turn would represent successful deployment and acceptance if traffic is above one hundred per month.

Challenges

The **pAPI** application is an innovative solution for modern product that is growing steadily in necessity, and relevance.

The Key Challenges can be broken down by Module.

Front-End

- Learning React JS
- Limitations of First tier Mongo DB, Heroku, AWS
- Using Best Practice
- Integration

Back-End

- Learning Multiple Languages, Environments, and Frameworks
- Using a NoSQL database
- Using Best Practice
- Network reliability, sustainability, manageability.
- Security

Miscellaneous

- Skill Levels of the team
- Time frames

All of the challenges were overcome with persistence and dedication from the team.

Summary

pAPI will create and generate a user-friendly API library that will provide an easy-to-use interface where customers/Users can browse for and utilise APIs for their Applications.

Simply the complex world of the modern API is made easier with a one stop shop for the discovery of useable APIs for the plethora of applications being developed.

pAPI will use the MERN technology stack, robust testing, and transparent methodology to develop a highly relevant product for a fast-growing market. The objective will be to educate and facilitate easy API integration.

Appendices.

All versions with full code support and ReadMe documentation are available from https://github.com/oliverschweikert/pAPI.