International Journal of Advanced Science and Technology
Vol. 30, No. 1, (2021), pp.117-138

Application Programming Interface (API) Library for Nigerian Software Development Industry

Mbato, Nnwobuike Robinson[1] and Ojekudo, Nathaniel Akpofure (Ph.D)[2]

[1]Research Scholar, Department of Computer Science, Faculty of Natural and Applied Sciences, Ignatius Ajuru University of Education, Rumuolumeni, Port Harcourt, Rivers State, Nigeria

[2]Department of Computer Science, Faculty of Natural and Applied Sciences, Ignatius Ajuru University of Education, Rumuolumeni, Port Harcourt, Rivers State, Nigeria

ABSTRACT

The concept of re-using functions or routines (or sometimes called libraries) built by the developer or a third party to address critical problems within applications by the developer without re-inventing the wheel is not new in software development. However, there are some extremely difficult situations in which an application must communicate with one or more other applications in order to share data/information without having prior knowledge of how such data/information is derived; thus, the name of such a process is Application Programming Interface (API).In the Nigerian software development industry, developers who build applications to access national critical data/information on a daily basis attempt to re-event the wheels where such capacity exists at the expense of cost and time, and are frequently confronted with limitations due to a lack of publicly re-usable functions, routines, or libraries that will enable access to such national critical data. This paper will offer a solution to this problem by providing a framework and an implemented application boilerplate that will allow Nigerian software developers to gain secure access to dozens of APIs that may target national critical data/information such as BVN, NIN, biometrics data, security data from security agencies, customs, NDLEA, immigration, licensing authorities and any other national critical data/information required by an application programmer without re-inventing the wheel thereby saving cost, time and removing limitations currently suffered by developers. The exploratory and explanatory research methodologies, as well as the Angular framework, were used to examine the impact of not reinventing the wheel. This work may be useful to programmers, API developers, researchers, security agencies, and any organization tasked with managing national critical data.

Keywords: API, BVN, NIN, NDLEA

Introduction

The concept of re-using functions, routines, or libraries created by the developer or a third party to address critical problems within applications created by the developer without re-inventing the wheel is not new in software development. However, there are extremely difficult situations in which an application must communicate with one or more other applications in order to share data/information in order to address a specific challenge without knowing how such data/information shared. are derived through a process called Application Programming Interface (API). Developers in Nigeria's software development industry who build applications to access national critical data/information on a daily basis attempt to re-event the wheels when such capacity exists at the expense of cost and time, In most cases, limitations arise as a result of a lack of publicly re-usable functions, routines, or libraries that will allow access to such national critical information/data. This paper will provide a framework for a library of APIs that will assist the Nigerian software development industry in addressing the challenges associated with accessing national critical data/information. It will also draw on some of the work done by the Nigerian Identity Management Commission (NIMC) and other government agencies to provide API libraries.

An Application Programming Interface (API) enables multiple devices that are linked together to communicate intelligently and share resources when necessary. It enables software written by different people in different locations to work together seamlessly (Sagdeo, 2018). An API is a collection of routines, rules, and tools that are used to create software applications. It specifies the type of inputs required to access a specific function as well as the expected outputs as a result of that input. API defines functionalities that are independent of their implementations, allowing definitions and implementations to differ without compromising. A good API makes it easier for developers to create programs by providing all of the building blocks required by the developer. The blocks are then assembled by a programmer. For example, you may see the Facebook or Google buttons as alternative sign-in options in most sign-up forms for websites or applications. These forms were not created from scratch, but their APIs allowed their functionalities to be integrated into the app's sign-up page.

API’s are used for accessing databases or computer hardware, such as hard disk drives or video cards, which are not under the programmer's control or management. This feature allows applications to share data and other information, thereby expanding the functionalities and capabilities of their various offerings. This paper seeks to provide a framework that will Increase the capabilities of their offerings by implementing API libraries designed for the Nigerian software development industry.

Statement of the Problem

The Nigerian software development industry does not have a library of APIs that have been specifically developed to support their development needs, particularly when accessing national critical data/information Through the implementation of the National Identity Number, the National Identity Management Commission) provided NIMC Verification Service (NVS) platform to enable verification of identities of persons enrolled in the National Identity Database. This API does not provide support for the developers to also directly access other national critical data/information from other partner agencies or organization seamlessly. These partner agencies or organizations include:

- Independent National Electoral Commission for Voters Registration
- National Health Insurance Scheme for Health Insurance Owners
- National Population Commission for Census and birth/death Registry
- National Pension Commission for National Data Bank
- Nigerian Immigration Service for International Passports
- Federal Road Safety Commission for Drivers Licenses
- Federal Inland Revenue Service for Tax Payers Database (UTIN)
- Department of National Civic Registration for national Identity Cards
- Nigeria Communication Commission (NCC) for SIM card Registry
- The Nigerian Police force
- Nigeria Prison Services (NPS) now known as Nigeria Correctional Centre
- Joint Tax Board (JTB)
- Corporate Affairs Commission (CAC)
- Economic and Financial Crime Commission (EFCC)
- Central Bank of Nigeria (CBN)
- State Security Service (SSS)
- Office of the National Security
- Third Party Agent/Agency

Aim and Objectives

The aim of this study is to create a framework for API libraries that will help the Nigerian software development industry access national critical data/information needed to address software-specific needs or deliverables in the most cost-effective way possible. Furthermore, build an application that will serve as a boilerplate to software developers that will consume the API.

Scope of the Study

This study provides an overview of the National Identity Management Commission's (NIMC) work in implementing the National Identity Number (NIN). Create an API library framework to aid the Nigerian software development industry. Create and deploy an application that will serve as a template for developers.

Significance of the Study

A framework of API libraries for the Nigerian software development industry to access critical national data/information will promote interaction between applications and allow software developers to access functions that they could not have developed themselves or that would have taken a significant amount of time to develop. This underscores the importance of the API libraries. This study could be beneficial to Nigerian software development industry, API developers, Government agencies, investors and researchers working to take this initiative to the next level.

Definition of Terms

- **i.** **Angular:** An Agile development environment powered by Google
- **ii.** **API:** this is an interface that defines the interaction between multiple software applications or mixed software-hardware intermediaries.
- **iii.** **API Library:** this is a collection of software protocols that a researcher can access through contract or agreement.
- **iv.** **Abstraction:** in OOP, this refereeing to hiding the unnecessary details of an application from the users and displaying only the essential attributes.
- **v.** **Polymorphism:** this is the provision of a single interface for entities of different types or the use of a single symbol to represent multiple different types.
- **vi.** **Inheritance:** this is a procedure in which a class inherits the attributes and methods of another class.
- **vii.** **Object Oriented Language:** this is a programming paradigm based on the concept of objects which can contain data and code.

Literature Review

Thousands of applications today communicate with one another via APIs to gain access to functionality outside of their own boxes in order to reposition their businesses for growth and profitability. Abigee (2016) stated in his report on the impact of APIs on digital business that "APIs are powering digital transformation across industries." To grow their digital businesses, enterprises are aggressively embracing an API-centric approach.

APIs and Library

An API is typically associated with a software library. The difference is that the API describes and prescribes expected behavior, whereas the library is an actual implementation of this set of rules. A single API may have multiple implementations (or none at all if it is abstract) in the form of different libraries that share the same programming interface. An API is also related to a software framework: a framework can be based on several libraries implementing several APIs, but unlike normal API use, access to the framework's behavior is mediated by extending its content with new classes plugged into the framework itself. Furthermore, through inversion of control or a similar mechanism, the overall program flow of control can be taken out of the caller's hands and placed in the hands of the framework

An API links a library to an application and is made up of two main parts:

a. Its declaring code, and
b. Its structure, sequence, and organization.

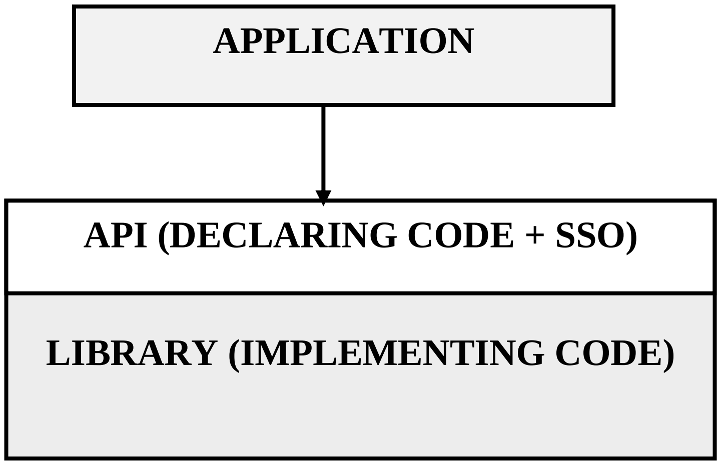


Figure 1: The Relationship between an Application, an API, and a Library (Sagdeo, 2018)

Declaring Code: This is the code that a developer who is using an API needs to invoke certain library functions. The declaring code is organized according to the sequence and organization taxonomy (“SSO”). An application can use an API without knowing anything about the underlying library's implementation as long as it adheres to the API's declaring code and SSO.A library, on the other hand, can be implemented in any way the implementer desires as long as it adheres to the API's SSO and declaring code.

And, while an API and a corresponding library are frequently created by the same organization, they are, in theory, completely separate works.

Software Libraries as a Means of Abstraction

The majority of software developed may be less useful if used in isolation. It is critical that they be integrated with other software in order for their utility to be recognized. To function optimally, some software relies on the functionalities provided by a library. For example, a certain prewritten code with well-defined inputs can be used to carry out a number of related functions (a clear example is the operating system libraries for android, Mac etc. which allows a program to communicate over a network, access storage devices and customize what the computer displays as an output).

Languages such as Java and Python provide a standard library that allows programs written in those languages to access the operating system of a computer, as well as building blocks that implement commonly used functionalities. In MATLAB, for example, there are several toolboxes such as the image processing toolbox and the computer vision toolbox that reduce coding to just a form of GUI or interface to achieve the same feat.

Libraries boost the speed and dependability of application development. They assist programmers in achieving a high level of abstraction, i.e. developing an application using several building blocks rather than starting from scratch. This reduces the complexity of the development process.Application developers can create high-level software that is not concerned with specific methods of computation or the manner in which the hardware must be used to achieve a specific result by using libraries built on top of libraries. This is an extremely powerful abstraction because it enables, for example, the rapid development of an Android application to run on thousands of hardware configurations (Sagdeo, 2018).

Language Bindings and Interface Generators

Arguments for APIs that will be used by more than one high-level programming language are designed using facilities that automatically map the API to features (syntactic or semantic) that are more natural in those languages. This is known as language binding, and it is an API in and of itself. The goal is to encapsulate the majority of the API's required functionality while leaving a "thin" layer appropriate for each language. Some interface generator tools that bind languages to APIs at compile time are listed below:

i. **SWIG**: an open-source interfaces bindings generator supporting numerous programming languages.
ii. **F2PY:** a Fortran to Python interface generator

Web APIs

Types of APIs and API protocols

APIs are classified into types based on what they are designed to deliver, as well as the protocols and standards that go with them.

- **Open APIs**

This type of API is also known as a public API because it is freely accessible to developers and other users. They may require registration before use, or they may require an API Key or OAuth to access them, or they may be completely open, depending on the owner's perspective (developer). The main focus of this type of API is for external users to access data or services.

- **Partner APIs**

These are APIs made available by/to strategic business partners in order to achieve a common goal. Because of the nature of the data or services they provide, they are not open to the public and require special permission to access. They are typically exposed to a public API developer portal, which developers can use in self-service mode with an on boarding process. This type of API is provided by NIMC via its Enhanced NIMC Verification System API.

- **Internal APIs**

This type of API is also referred to as a private API because it is hidden from external users and is only accessible through internal systems. The APIs are not intended for use outside of the company or the team that created them, but rather for use across various internal development teams to improve productivity and reuse services to save time and money.

- **Composite APIs**

This API combines several data or service APIs. They are built by utilizing an API creation tool's API orchestration capabilities to create a sequence of tasks that run synchronously as a result of the execution. They enable developers to access multiple endpoints in a single call.

API architecture and protocols (Web services)

There are established guidelines for making API calls. Protocols are the names given to these predefined rules. Different API architectures specify different rules for consuming the API, as well as constraints. The rules define the data types and commands that can be used with a given API.

- **REST (Representational State Transfer**)

REST entered the API space later and is still very popular web API architecture. To qualify as REST, an API must adhere to the architectural principles or constraints outlined below.

i. **Client-Server Architecture**: the interface is separated from the backend and data storage. This allows for flexibility, and for different components to evolve independent of each other.
ii. **Statelessness**: No client context is stored on the server between requests, challenge/constraint that an API consumer must manage to get a desired result.
iii. **Cacheability**: Clients can cache responses, so a REST API response must explicitly state whether it can be cached or not.
iv. **Uniform Interface**: A client and server should communicate with one another via HTTP (HyperText Transfer Protocol) using URIs (Unique Resource Identifiers), CRUD (Create, Read, Update, Delete), and JSON (JavaScript Object Notation) conventions.
v. **Layered system**: The API will work whether it is communicating directly with a server, or through an intermediary such as a load balancer.

- **SOAP**

The SOAP (Simple Object Access Protocol) protocol is a well-known web API protocol. It is designed to be extensible, neutral (capable of operating over a variety of communication protocols such as HTTP, SMTP, TCP, and others), and independent (it allows for any programming style).

The SOAP specification is as represented below:

a. The processing model: how to process a SOAP message.
b. Extensibility model: SOAP features and modules.
c. Protocol binding rules: how to use SOAP with an underlying protocol, such as HTTP.
d. Message construct: how to structure a SOAP message.

SOAP and REST are two of the most common structures for designing web APIs, and they are of interest in this study. Both define a standardized structure for sending and receiving requests and responses via web APIs. It ignores the semantics of the underlying protocol, which is typically served as HTTP [5].

The most common formats used in web APIs for data exchanged via APIs are JavaScript Object Notation (JSON) and Extensible Markup Language (XML) (XML). JSON is an open-standard, lightweight message format that uses a key-value pair to represent data. Unlike JSON, XML structures data using nodes. Clients can select which format is available to them by specifying it in the HTTP request.

Because HTTP is so widely used, many studies focus on web API issues. However, only clients can initiate requests to servers in interactions between an HTTP server and a client via web APIs.

Review of Related Work

Some researchers have done work in the field of API design for the software development industry, some of which are listed below:

Sagdeo (2018) discussed application programming interfaces and the problem of standardization-value appropriation. The study separated the value of an API copyright into two parts: the API's expression value and its standardization value. The standardization value of an API is essentially the user switching costs that an API owner can appropriate through the right to exclude. He argued that appropriation of standardization value is not required to incentivize innovation and, in fact, causes many of the social harms that commentators have long associated with copyright on APIs. Four legal regimes were discussed that prevented an API's owner from appropriating its standardization value and concluded with a variable-rate compulsory licensing regime.

Obajemu et al. (2013) presented a survey on the utilization and evaluation of library products in Nigeria. The study's goal was to raise awareness of existing software in Nigeria in order to improve selection quality. The study also provided practical steps to take when selecting a library product and identified operational issues with library software. A total of 50 Nigerian libraries were surveyed, including 11 polythenic libraries, three C.O.E libraries, and 22 federal, state, and private libraries. The survey data was gathered using 120 structured questionnaires. The findings revealed that the majority of the correspondents agreed on the practical steps for acquiring library software.

Hou et al. (2016) proposed the design and implementation of an Application Programming Interface for the cloud of the internet of things. Considering the large number of resource-constrained IoT devices, they proposed various types of APIs with various application protocols. Finally, they conducted a series of experiments to evaluate the designed APIs, and the results were analyzed.

Santoro et al. (2019) proposed a report on web APIs. The report was created to assist API stakeholders in identifying and selecting API solutions. A series of definitions were provided to provide clarity on concepts, standards, and technical specifications of API. The description and classification of the collected standards were also presented. A shortlist of these documents is also proposed, along with a brief description of each of them (based on their utilization, maintenance, and stability). A collection of API-related European Commission creations was also presented. Finally, the report provided valuable appendix with definitions of the pertinent footings collected within the APIs4DGov study.

Meng et al. (2018) provided API documentation. The findings of several semi-structured interviews and a follow-up questionnaire conducted to investigate software developers' learning goals and learning strategies, including the information resources they use and the quality criteria they apply to API documentation, were presented. According to their findings, developers initially attempted to form a global understanding of an API's overall purpose and main features, but then adopted either a concepts-oriented or a code-oriented learning strategy that API documentation both needed to address. Furthermore, general quality criteria such as completeness and clarity were applicable to API documentation.

Lutters and Seaman (2007) investigated the use of documentation in software maintenance environments by using storytelling as a data collection technique. According to their results, structural properties of documentation (e.g., tables of contents and indices) were crucial with respect to the ability of maintainers to make use of it.

NIMC (https://nimc.gov.ng/contact/) provided API's for developers to access national critical data/information. The challenge with their API is that you cannot use it to access other partner agencies data/information seamlessly.

Research Methodology

In this work, explanatory and exploratory research methodologies are used to examine the impact of not reinventing the wheel.

Discussion

Lack of availability and accessibility of a reliable national critical data/information to help address myriads of issues and challenges that have plagued the Nigerian nation is not new. The Nigerian government through the establishment of the National Identity Management Commission (NIMC) is implementing a National Identity Number (NIN) scheme that will harmonize some aspect of national critical data/information and make them available to government agencies and other partners through the implementation of their Web services API, enhanced NIMC Verification System (eNVS). This API enables verification of identities of persons enrolled in the National Identity Database through the exposure of a SOAP based web service which can be integrated into client applications. This API, however, does not integrate with other agencies and partners databases to get national critical data/information that they processed and kept. This leaves the application developer with a limited option on how to get some national critical data/information to address a particular concern.

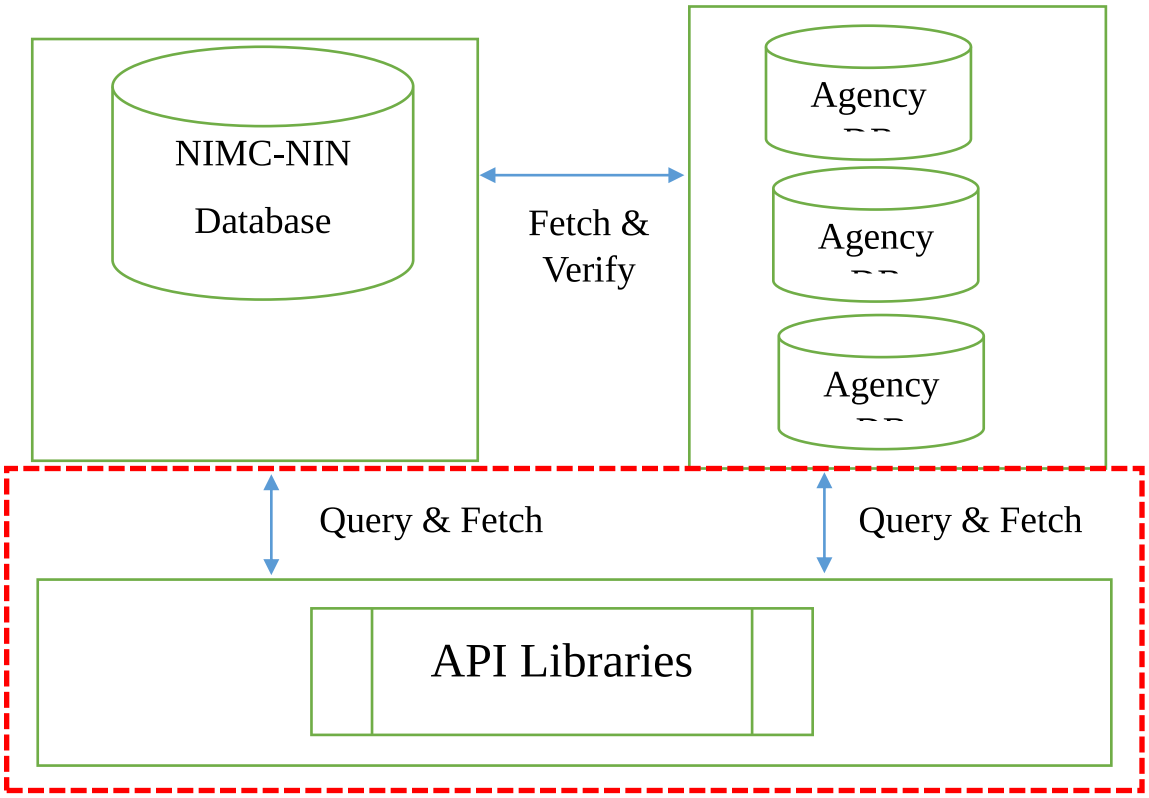


Fig 2: Architecture of API Collaborative Framework

Our Approach

We will develop a framework of API libraries that will act as a one stop shop to application developers to access national critical data/information. In addition, we will also develop a boilerplate computer program that will guide the implementation of the API libraries as a case study or proof of concept using the Nigerian Police Force and National Identity Management Commission's National Identity Number (NIN) simulated databases. We have identified 17 partner agencies/commissions of government and a third-party agency that could also be integrated into the API Libraries.

It is important to note that how agencies/commissions keep and manage their databases is beyond the scope of this study. This study assumes that these databases exist and that the national critical data/information that are kept by these agencies/commissions are accessible through our API through the interfaces they provide. However, a framework could also be built that can aggregate all national critical data/information into a common and centralized database thereby taking away the burdens of managing these databases to be available centrally at all times from the agencies/commissions.

From Figure 2, an agent calls the NIMC-NIN database to verify that the individual of interest exists in the national database and fetches any aspect of the persons demographic data or any other data that is relevant to the query and updates the agent's database accordingly. The application programmer through the API libraries queries the NIMC-NIN database or any partner/agent database to retrieve national critical data/information such as the persons crime record from the Nigerian Police force (NPF) database or tax record from the Federal Inland Revenue Service (FIRS) or any other agency/commission of interest.

The area bordered by broken red lines in fig 2 is new and this is the additional functionality that will be added/integrated into the API that the NIMC's NIN provides. This new functionality could enable the availability of national critical data/information for national security and growth of the Nigerian economy.

API Library Structure (Documentation)

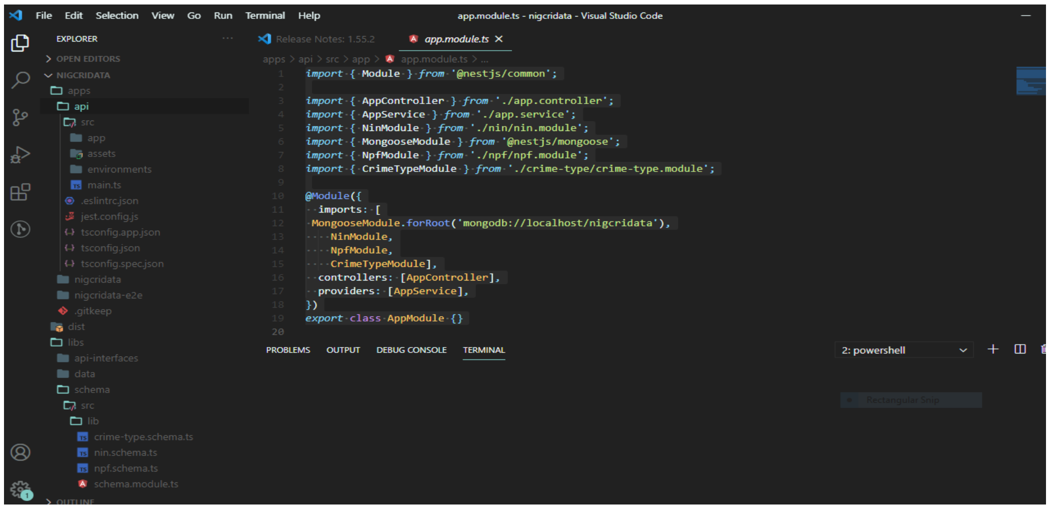
The proposed API library documentation is as shown in table 3.

Table 3
Proposed API Library Documentation

API Ref Code	**API Name**	**Description**
INEC	Independent National Electoral Commission	Voters Registration
NHIS	National Health Insurance Scheme	Health Insurance
NPC	National Population Commission	Census birth and death registry
PENCOM	National Pension Commission for	National Pension Data Bank
NIS	Nigerian Immigration Service for	International Passports
FRSC	Federal Road Safety Commission	Drivers Licenses
FIRS	Federal Inland Revenue Service	Tax payers Database (UTIN)
DCR	Department of National Civic Registration	National Identity Card
NCC	Nigeria Communication Commission	SIM card Registration
NPF	The Nigerian Police force	Crimes and offences
NPS	Nigeria Prison Services (NPS)	Database of prisoners
JTB	Joint Tax Board	Database of board members
CAC	Corporate Affairs Commission (CAC)	Company Registration

EFCC	Economic and Financial Crime Commission (EFCC)	Economic and financial crime
CBN	Central Bank of Nigeria	Banks and transactions
SSS	State Security Service	Secrete Services
ONS	Office of the National Security	National Security
TPA	Third Party Agent/Agency	Agencies

Table 3 depicts the structure of the API documentation for easy reference. It could be a work in progress document for API developers that will develop this concept.



```
import { Module } from '@nestjs/common';

import { AppController } from './app.controller';
import { AppService } from './app.service';
import { NinModule } from './nin/nin.module';
import { MongooseModule } from '@nestjs/mongoose';
import { NpfModule } from './npf/npf.module';
import { CrimeTypeModule } from './crime-type/crime-type.module';

@Module({
  imports: [
 MongooseModule.forRoot('mongodb://localhost/nigcridata'),
    NinModule,
    NpfModule,
    CrimeTypeModule],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Figure 3: Sample Code Snippet

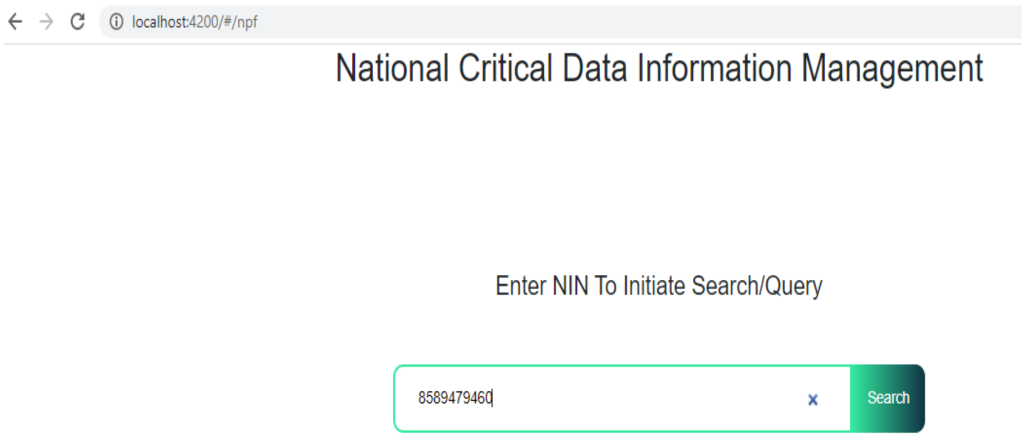
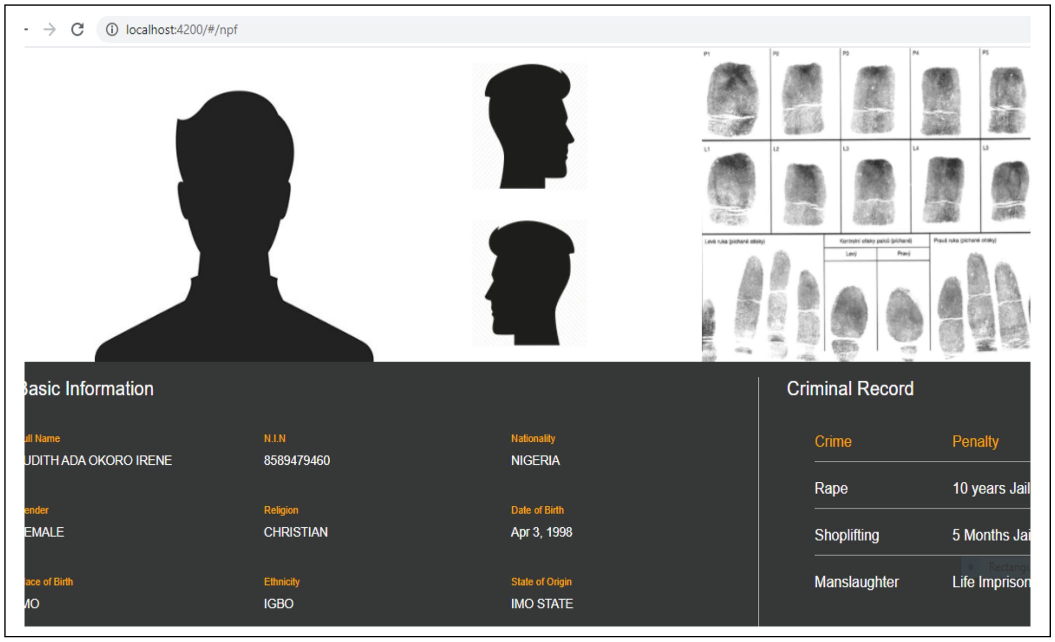

localhost:4200/#/npf
National Critical Data Information Management
Enter NIN To Initiate Search/Query
8589479460
Search

Fig. 4: Query Screen

API Library Implementation

The API library is built with the Angular development tool, which is compatible with the Agile Methodology framework. This implementation or framework could serve as a starting point for API developers. Appendix A contains a list of the codes for the Main Modules (back and front ends), Methods, data models, views, and so on. MongoDB is the database used in the implementation.



3asic Information

ıll Name	N.I.N	Nationality
UDITH ADA OKORO IRENE	8589479460	NIGERIA
ender	Religion	Date of Birth
EMALE	CHRISTIAN	Apr 3, 1998
lace of Birth	Ethnicity	State of Origin
MO	IGBO	IMO STATE

Criminal Record

Crime	Penalty
Rape	10 years Jail
Shoplifting	5 Months Jai
Manslaughter	Life Imprison

Fig. 5: Sample Report Screen

Conclusion

Fighting crime, criminality, and other social vices is essential if Nigeria is to solve her current security challenges and grow her economy. To win this war, it is critical to provide and share national critical data/information among the agencies/commissions that require such data/information to fight the war. We have provided a framework of API libraries to assist the Nigerian Software Development Industry in gaining access to such national critical data/information that will aid various agencies/commissions. This API libraries will save developers money while also removing current limitations and challenges. We also implemented an API library that will serve as a boilerplate.

References

1. Customer Information Manager (CIM). (2013). SOAP API Documentation. *Authorize.Net*. JUL. 2013.

2. Hou, L., Zhao, S., Li, X., Chatzimisios, P. and Zheng, K. (2016). Design and Implementation of Application Programming Interface. *International Journal of Network Management*. 1-16. DOI: 10.1002/nem.1936.

3. Lin B, Chen Y, Chen X, Yu Y. (2013). Comparison between JSON and XML in applications based on AJAX. *International Conference on Computer Science & Service System (CSSS)*. 1174–1177.

4. Lutters, W. G. and Seaman, C. B. (2007). Revealing actual documentation usage in software maintenance through war stories. *Information and Software Technology*, 49(6), 576–587.

5. Martin. F. (2018). Inversion of Control. *International Journal of Engineering and Research*. 3(1) 111-115.

6. Meng, M., Steinhardt, S. and Schubert, A. (2018). Application Programming Interface Documentation: What do Software Developers Want? *Journal of Technical Writing and Communication.* 48(3). 295-330.

7. Mohamed. F. (2016). Object-Oriented Application Frameworks. *International Journal of Computer Science and Engineering*. 7(4), 10-14.

8. Obajemu, A. S. Osagie, J. N., kinade, H. O. J. and. Ekere, F. C. Library Software Products in Nigeria: A Survey of Uses and Assessment. *International journal of Library and Information science.* 5(5). 113-125.

9. Santoro, M. Vaccari, L. Mavridis, D. Smith, R. Posada, M. and Gattwinkel,D. (2019). Web APIs: general-purpose standards, terms and European Commission initiatives. EUR 29984 EN, Publications Office of the European Union, Luxembourg, 2019, ISBN 978-92-76-13183-0, doi: 10.2760/675, JRC118082.

10. Sagdeo. P. (2018). Application Programming Interface and the Standardization-Value Appropriation Problem. *Harvard Journal of Law and Technology*. 32(1). 237-255.

Appendix A

SOURCE CODE

Main Module (API)--Backend

```
import { Module } from '@nestjs/common';

import { AppController } from './app.controller';
import { AppService } from './app.service';
import { NinModule } from './nin/nin.module';
import { MongooseModule } from '@nestjs/mongoose';
import { NpfModule } from './npf/npf.module';
import { CrimeTypeModule } from './crime-type/crime-type.module';

@Module({
  imports: [
 MongooseModule.forRoot('mongodb://localhost/nigcridata'),
    NinModule,
    NpfModule,
    CrimeTypeModule],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Main Methods: National Identity Number (NIN)

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Nin } from '@nigcridata/schema';
import { Model } from 'mongoose';
import * as randomize from "randomatic";

@Injectable()
export class NinService {

  constructor(
    @InjectModel(Nin.name)
    private readonly ninModel: Model<Nin>,
  ) {}

  async getAllNin(): Promise<Nin[]> {
    return await this.ninModel.find().exec();
  }

  async getNin(id: string): Promise<Nin> {
    return this.ninModel.findById({_id: id}).exec();
  }


  async createNin(nin: Nin): Promise<Nin> {
    const NinNumber = randomize("0", 11);
    const parseValue = +NinNumber
    console.log('value ', parseValue)
    const newNin = {
      ...nin,
      NinNumber: +NinNumber,
    }
   const savedNin = await this.ninModel.create(newNin);
   return savedNin.save();
  }

  async updateNin(nin: Nin): Promise<Nin> {
   return this.ninModel.findOneAndUpdate({ _id: nin['_id'] }, {$set: nin}, {new: true}).exec();
  }

  async deleteNin(id: string) {
    const found = await this.getNin(id);
    if(!found) {
      throw new NotFoundException('nin not found');
    }
    await this.ninModel.findByIdAndDelete(id);
  }
}
```

Main Methods: Nigeria Police Force (NPF)

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { NigeriaPoliceForce, Nin } from '@nigcridata/schema';
import { Model } from 'mongoose';
import * as randomize from "randomatic";

@Injectable()
export class NpfService {

  constructor(
    @InjectModel(NigeriaPoliceForce.name)
    private readonly npfModel: Model<NigeriaPoliceForce>,
    @InjectModel(Nin.name)
    private readonly ninModel: Model<Nin>,
  ) {}

  async getAllNigeriaPoliceForce(): Promise<NigeriaPoliceForce[]> {
    return await this.npfModel.find().populate('crimes').exec();
  }

  async getNigeriaPoliceForce(id: string): Promise<NigeriaPoliceForce> {
    return await this.npfModel.findById({_id: id}).populate('crimes').exec();
  }

  async searchNigeriaPoliceForce(nin: number): Promise<NigeriaPoliceForce> {

    const foundNin = await this.ninModel.findOne({NinNumber: nin}).exec();

    const foundNpf = await this.npfModel.findOne({NinNumber: nin}).populate('crimes').exec();
    const transformData: any = {
      npf: foundNpf,
      nin: foundNin
    }
    return transformData;
  }


  async createNigeriaPoliceForce(npf: NigeriaPoliceForce): Promise<NigeriaPoliceForce> {
    const OffenderCode = randomize("0", 6);
    const newNpf = {
      ...npf,
      OfferderCode
    }
   const savedNigeriaPoliceForce = await this.npfModel.create(newNpf);
   return (await savedNigeriaPoliceForce.save()).populate('crimes');
  }

  async updateNigeriaPoliceForce(npf: NigeriaPoliceForce): Promise<NigeriaPoliceForce> {
   return this.npfModel.findOneAndUpdate({ _id: npf['_id'] }, {$set: npf}, {new: true}).exec();
  }

  async deleteNigeriaPoliceForce(id: string) {
    const found = await this.getNigeriaPoliceForce(id);

    if(!found) {
      throw new NotFoundException('npf not found');
    }
    await this.npfModel.findByIdAndDelete(id);
  }
}
```

Main Methods: Crime Type

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { CrimeType } from '@nigcridata/schema';
import { Model } from 'mongoose';

@Injectable()
export class CrimeTypeService {
  constructor(
    @InjectModel(CrimeType.name)
    private readonly crimeTypeModel: Model<CrimeType>,
  ) {}

  async getAllCrimeType(): Promise<CrimeType[]> {
    return await this.crimeTypeModel.find().exec();
  }

  async getCrimeType(id: string): Promise<CrimeType> {
    return this.crimeTypeModel.findById({_id: id}).exec();
  }


  async createCrimeType(crimeType: CrimeType): Promise<CrimeType> {
   const savedCrimeType = await this.crimeTypeModel.create(crimeType);
   return savedCrimeType.save();
  }
  async updateCrimeType(crimeType: CrimeType): Promise<CrimeType> {
   return this.crimeTypeModel.findOneAndUpdate({ _id: crimeType['_id'] }, {$set: crimeType}, {new:
true}).exec();
  }

  async deleteCrimeType(id: string) {
    const found = await this.getCrimeType(id);

    if(!found) {
      throw new NotFoundException('crime type not found');
    }
    await this.crimeTypeModel.findByIdAndDelete(id);
  }
}
```

Main Module (Front End)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { HttpClientModule } from '@angular/common/http';
import { SpinnersModule } from '@nigcridata/spinners';
import { AppRoutingModule } from './app-routing.module';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { NpfContainerComponent } from './containers/npf-container/npf-container.component';
import { NpfComponent } from './components/npf/npf.component';
import { SearchComponent } from './components/search/search.component';
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent,
    NpfContainerComponent,
    NpfComponent,
    SearchComponent,
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    HttpClientModule,
    AppRoutingModule,
    ReactiveFormsModule,
    SpinnersModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Search (TS) Type Script

```
import { Component, ElementRef, EventEmitter, OnInit, Output, ViewChild } from '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
  // eslint-disable-next-line @angular-eslint/component-selector
  selector: 'app-search',
  templateUrl: './search.component.html',
  styleUrls: ['./search.component.scss']
})
export class SearchComponent {

  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  @Output() searchEmit = new EventEmitter<any>();
  @ViewChild('search', { static: true }) searchForm: ElementRef;
  control: FormControl = new FormControl();


  onSubmit() {
    this.searchEmit.emit(this.control.value)
  }

}
```

Search html

```
  <div class="the_search">

    <h2 class="app_title">National Critical Data Information Management</h2>
    <div class="" style="width:100%; display: flex; justify-content: center; flex-direction: column; align-
items: center;">
      <p class="tip">Enter NIN to initiate search/query</p>
    <div class="search_container">
      <div class="search_input">
        <i class="icofont-search"></i>
        <input type="search" class="search" [formControl]="control" placeholder="Enter NIN here" />
      </div>
      <button class="submit_search" (click)="onSubmit()">Search</button>
    </div>
    </div>
  </div>
```

Display Record (TS)

```
import { Component, EventEmitter, Input, OnChanges, Output, SimpleChanges } from '@angular/core';
import { NigeriaPoliceForceDto } from '@nigcridata/data';

@Component({
  // eslint-disable-next-line @angular-eslint/component-selector
  selector: 'app-npf',
  templateUrl: './npf.component.html',
  styleUrls: ['./npf.component.scss']
})
export class NpfComponent implements OnChanges {

  @Input() npf: NigeriaPoliceForceDto;
  @Output() taskEmit = new EventEmitter<any>();

  ngOnChanges(changes: SimpleChanges): void {
    //Called before any other lifecycle hook. Use it to inject dependencies, but avoid any serious work he
re.
    //Add '${implements OnChanges}' to the class.
    console.log('found ', this.npf);
    if(this.npf) {

      this.taskEmit.emit('list')
    }
  }

}
```

Display Record HTML

```
 <button class="back" (click)="taskEmit.emit('search')">Back</button>
<div class="profile" >


  <ng-container *ngIf="npf?.nin; else noNpf">
  <div class="primary_section">
    <div class="passport_views">
      <img src="assets/images/Male-Profile-Picture.jpg" alt="" class="passport" />
      <div class="other_views">
        <img src="assets/images/16-512.webp" alt="" class="right" />
        <img src="assets/images/16-512.webp" alt="" class="left" />
      </div>
    </div>
    <img src="assets/images/426317_1_En_1_Fig5_HTML.png" alt="" class="prints" />
  </div>

  <div class="other_information">
    <div class="info_container">
      <h2 class="section_title">Basic Information</h2>
```

```
   <div class="basic_info">
     <p class="info">
       <span class="tag">Full Name</span>
       <span class="answer">{{npf?.nin?.FirstName}} {{npf?.nin?.MiddleName}} {{npf?.nin?.SurNa
me}} {{npf?.nin?.MaidenName}}</span>
     </p>
     <p class="info">
       <span class="tag">N.I.N</span>
       <span class="answer">{{npf?.nin?.NinNumber}}</span>
     </p>
     <p class="info">
       <span class="tag">Nationality</span>
       <span class="answer">{{npf?.nin?.Nationality}}</span>
     </p>
     <p class="info">
       <span class="tag">Gender</span>
       <span class="answer">{{npf?.nin?.Gender}}</span>
     </p>
     <p class="info">
       <span class="tag">Religion</span>
       <span class="answer">{{npf?.nin?.Religion}}</span>
     </p>
     <p class="info">
       <span class="tag">Date of Birth</span>
       <span class="answer">{{npf?.nin?.DateOfBirth | date}}</span>
     </p>
     <p class="info">
       <span class="tag">Place of Birth</span>
       <span class="answer">{{npf?.nin?.PlaceOfBirth}}</span>
     </p>
     <p class="info">
       <span class="tag">Ethnicity</span>
       <span class="answer">{{npf?.nin?.Ethnicity}}</span>
     </p>
     <p class="info">
       <span class="tag">State of Origin</span>
       <span class="answer">{{npf?.nin?.StateOfOrigin}}</span>
     </p>
     <p class="info">
       <span class="tag">Local Government Area</span>
       <span class="answer">{{npf?.nin?.LgaOfOrigin}}</span>
     </p>
     <p class="info">
       <span class="tag">Height</span>
       <span class="answer">{{npf?.nin?.Height}}</span>
     </p>
     <p class="info">
       <span class="tag">color of Hair</span>
       <span class="answer">{{npf?.nin?.ColorOfHair}}</span>
     </p>
```

```
        <p class="info">
          <span class="tag">color of Eye</span>
          <span class="answer">{{npf?.nin?.ColorOfEye}}</span>
        </p>
      </div>
    </div>

    <div class="criminal_record">
      <h2 class="section_title">Criminal Record</h2>

      <ul class="record_list">
        <li class="criminal_act" style="color: orange">
          <span>Crime</span>
          <span>Penalty</span>
          <span>Status</span>
        </li>
        <li class="criminal_act" *ngFor="let crime of npf?.npf?.crimes">
          <span>{{crime?.Crime}}</span>
          <span>{{crime?.Penalty}}</span>
          <span>{{crime?.Status}}</span>
        </li>


      </ul>
    </div>
  </div>
</ng-container>
</div>


<ng-template #noNpf>
<div class="alert alert-danger text-center">No record found</div>
</ng-template>
```

Container (Search & View Record) TS

```
import { Component, OnInit } from '@angular/core';
import { NigeriaPoliceForceDto } from '@nigcridata/data';
import { Observable } from 'rxjs';
import { NpfService } from '../../services/npf.service';

@Component({
  // eslint-disable-next-line @angular-eslint/component-selector
  selector: 'app-npf-container',
  templateUrl: './npf-container.component.html',
  styleUrls: ['./npf-container.component.scss']
})
export class NpfContainerComponent {
  npf$: Observable<NigeriaPoliceForceDto>;
  ViewTypeOption = 'search';
  constructor(public npfService: NpfService) { }



  onSearch(value) {

    const foundNpf$ = this.npfService.searchOffender(+value);
    this.npf$ = this.npfService.showLoaderUntilComplement(foundNpf$);
    this.ViewTypeOption = 'list';
  }

  onTask(value) {
    console.log('found 1 ', value);
    this.ViewTypeOption = value;
  }

}
```

Container (search & record) html

```

<main class="main_page">
  <div [ngSwitch]="ViewTypeOption">

    <ng-template ngSwitchCase="search">
<app-search (searchEmit)="onSearch($event)"></app-search>
</ng-template>
<ng-container *ngIf="(npf$ | async) as npf">
<ng-template ngSwitchCase="list">

<app-npf
[npf]="npf"
(taskEmit)="onTask($event)"
*ngIf="(npfService.isLoadingSite$ | async) === false"
></app-npf>
</ng-template>
</ng-container>
</div>
<cradua-list-spinner
*ngIf="(npfService.isLoadingSite$ | async) === true"
>
</cradua-list-spinner>
</main>
```