Software Design Description

1. Frontspiece
    1.1. Date of issue and status
        1.1.1. October 1st, 2018, revision A completed
    1.2. Issuing organization
        1.2.1. Team Begone Router
    1.3. Authorship
        1.3.1. Klaus Cipi
        1.3.2. Peter Banis
        1.3.3. Michael Kolar
        1.3.4. Robert Olsen
    1.4. Change history
        1.4.1. September 25 : initial draft started
        1.4.2. September 28: Update: added stakeholders
        1.4.3. September 28: Added Diagrams
        1.4.4. September 29: Update: Concerns: Responsiveness, Reliability, Functionality
        1.4.5. September 29: Update: Design View
2. Introduction
    2.1. Purpose
        2.1.1. This document describes the design of the ad-hoc networking API, the P2P Android extension and the installation verification tool that accompanies the API. It shall cover the organization of functions into classes and packages, and how the communication between the Ad-Hoc protocol and P2P protocol will be handled.
    2.2. Scope
        2.2.1. This document takes all requirements specified in the SRS and describes how they will be implemented.
        2.2.2. This document contains a complete description of Java API for Ad-hoc and P2P networking.
        2.2.3. The basic architecture is a repository of Java files alongside some scripting files for the installation verification tool.
    2.3. Context
        2.3.1. This project serves as an extension to the pre-established P2P method between the Windows, MacOS, and Linux platforms. There is no current means of interconnection via the ad-hoc model. This document details the design process through which the ad-hoc model is established amongst these platforms, as well as the integration of Android-based P2P and ad-hoc modules.
    2.4. Summary
        2.4.1. This document illustrates how the API will unify the main operating systems (Windows, MacOS, Linux, and Android) through expanding on P2P capabilities, promoting backwards compatibility via ad-hoc networking, and ensuring ease of setup with an

installation verification tool. These design detailings provide insight for the planned and executed approaches to constructing the internal architecture of the API, and will conform to the standards set forth in IEEE 1016-2009. The intended audience of this design document mainly consists of the software engineers who contribute to and maintain the system, but does not exclude those who are otherwise interested in its internals.

3. References

3.1. This document shall adhere to the following publications:

3.1.1. IEEE 1016-2009, IEEE Standard for Information Technology—Systems Design—Software Design Descriptions

4. Glossary

5. Body

5.1. Identified stakeholders and design concerns

5.1.1. Stakeholders

5.1.1.1. API providers

5.1.1.1.1. API providers are people who design, build, and distribute the API. Often times, API providers are software engineers or developers that will provide the API system for distribution. Furthermore, API providers have the duty to maintain and update their API to keep it up-to-date with the current technology and make it more appealing to the API consumers and customers. Typically, API providers set up a portal or a system to showcase their API and communicate with the potential users. APIs shall be modular, efficient, and well documented.

5.1.1.2. API consumers (developers)

5.1.1.2.1. API consumers are software developers that shall integrate the API build by API providers into their own system or web application. For the most part, API consumers shall be considered as the most important stakeholders to the API providers because they will develop upon and use the API. Consumers look for well-written, well-documented, and easily to understand API.

5.1.1.3. API customer

5.1.1.3.1. API customers shall not be mistaken for API consumers. Customers are the ones that decide which API shall be used in their application and pay for the use of API. Usually, API customers purchase an API to solve a problem that they cannot solve because they either don’t have resources, data, or time. When purchasing an API, customers take into the account the efficiency and simplicity of the API.

These customers will commonly be companies that employee the developers which directly utilize the API.

5.1.1.4. API end-users

5.1.1.4.1. API end-users use the API indirectly and most of the time they are not aware that they are using a specific API. Generally, end-users deal with applications that are build upon a specific API by consumers that were mentioned above. Even though end-users are not directly using or accessing the API, they can be affected by the features that a particular API has. For example, security, performance, availability, and stability directly impacts the end-user. Depending on the end-user feedback, developers might decide to search for an alternative API or keep using the same one.

5.1.2. Concerns

5.1.2.1. Responsiveness?

5.1.2.1.1. API Consumers (developers)
API developers do not want to introduce a large technical overhead in wireless communication. They may already have notable constraints within their own application and so require the API to be quick to transfer, receive and process data as needed.

5.1.2.1.2. API customer (non-developers)
Companies that license the API want a competitive advantage in the marketplace. They know that their consumers will find value in quick and convenient wireless networking. This means that they will need to have a solid foundational code base that meet these needs if they are to stay relevant in their market. If the API allows for the aforementioned responsiveness benefits to their internal developers, then they will be ahead of their competitors.

5.1.2.1.3. API end-users
API end-users are everyday users of a software system utilizing the API. They may have constraints on what hardware they can utilize, including the physical capabilities of the device. For any application using the API, they will be greatly inconvenienced if the API poses a significant hardware burden on them.

5.1.2.2. Reliability?

5.1.2.2.1. API consumers (developers)
API consumers have a need for the API to transfer data without failure, and without delay. For the API to

meet their needs means the API cannot mishandle data, it cannot fail to transfer data and it cannot have failures in behavior without warning the consumers.

5.1.2.2.2. API customers (non-developers)

Companies know that their consumers which use network based applications will expect that the service is always available to them. Every second a service is down means lost profits to the company. In addition, any mishaps, such as unavailable internet service or errors in data transfer, will create a negative reputation for the company. The survival of any company is based on their consumer base's faith in their products/services, so they want to ensure that their products are consistently providing utility to satisfy demand. The more reliable the API is, then the easier it will be for the company's engineers to avoid mistakes which threaten the company's future.

5.1.2.2.3. API end-users

API end-users place an additional reliance on API consumers, and so any reliably concern a consumer has is shared by the end-users.

5.1.2.3. Functionality?

5.1.2.3.1. API consumers (developers)

API consumers expect the API to provide at least everything available in the standards for Ad-hoc networking and P2P networking. They also desire as great an overlap between these standards as possible, to allow Android devices to communicate with Windows/Mac/Linux devices seamlessly. They expect that all the possible functionality is made clear, and any functionality that cannot be bridged between the two standards is also made clear to them.

5.1.2.3.2. API customers (non-developers)

The organizations, usually companies, which will employee most of these developers also have concerns about functionality. The more intellectual heavy-lifting the API handles, the less developers will need to homebrew their functions and features. This will allow them to create more sophisticated applications, and enable them to meet deadlines. This increased productivity will help them develop more sophisticated products with shorter deadlines which will help their revenues. The more the API serves this end, the more these customers will be incentivized to license the API.

5.1.2.3.3. API end-users
End-users place an additional reliance on API consumers, and so all of the API consumer concerns also apply to them. End-users also have an expectation that the functionality provided by the API does not require any work on their part, for example by rooting a device. The API consumer may not share this concern, which makes it unique to the end-user.

5.2. Design viewpoint

5.2.1. API provider

5.3. Design view

5.3.1. Design concerns

5.3.1.1. API providers shall focus on building, testing and publishing the API

5.3.2. Design elements

5.3.2.1. Design entities that concern the API provider are

5.3.2.1.1. Classes

5.3.2.1.2. Data storages

5.3.2.1.3. Generic templates

5.3.2.1.4. Libraries

5.3.2.1.5. Scripts

5.3.2.2. Design attributes

5.3.2.2.1. SCRIPT_MACOS_GET_WLAN_INTERFACE

5.3.2.2.1.1. Script

5.3.2.2.1.2. Used to get interface information from a device that runs MacOS

5.3.2.2.2. SCRIPT_Win_GET_WLAN_INTERFACE

5.3.2.2.2.1. Script

5.3.2.2.2.2. Used to get interface information from a device that runs Windows 7

5.3.2.2.3. SCRIPT_LINUX_GET_WLAN_INTERFACE

5.3.2.2.3.1. Script

5.3.2.2.3.2. Used to get interface information from a device that runs Linux OS

5.3.2.2.4. Script_Linux_Connect_Suid_Ad-hoc

5.3.2.2.4.1. Script

5.3.2.2.4.2. Used to start an ad-hoc connection on devices that run Linux
NOTE: Similar script exist for MacOS and Windows

5.3.2.2.5. Script_Linux_Disconnect_Suid_Ad-hoc

5.3.2.2.5.1. Script

5.3.2.2.5.2. Used to drop an ad-hoc connection on devices that run Linux

NOTE: Similar script exist for MacOS and Windows

- 5.3.2.2.6. Net.ddp2p
  - 5.3.2.2.6.1. Library
  - 5.3.2.2.6.2. Library used by providers to build the API because it provides functions to communicate with the hardware (ex. Ad-hoc wifi adapter)
- 5.3.2.2.7. BroadcastClient
  - 5.3.2.2.7.1. Class
  - 5.3.2.2.7.2. Use to broadcast information from a client to all the connected peers.
- 5.3.2.2.8. BroadcastData
  - 5.3.2.2.8.1. Class
  - 5.3.2.2.8.2. A data structure that will hold a list of interface names and DatagramSockets
- 5.3.2.2.9. BroadcastInterface
  - 5.3.2.2.9.1. Class
  - 5.3.2.2.9.2. A data structure that will hold broadcast interface information
- 5.3.2.2.10. BroadcastServer
  - 5.3.2.2.10.1. Class
  - 5.3.2.2.10.2. Used to receive information send by the client
- 5.3.2.2.11. Detect_interface
  - 5.3.2.2.11.1. Class
  - 5.3.2.2.11.2. This class detects which OS and initial SSIDs a device is using.
- 5.3.2.2.12. InterfaceData
  - 5.3.2.2.12.1. Class
  - 5.3.2.2.12.2. A data structure that holds basic information which is necessary to interfacing.
- 5.3.2.2.13. IPv4Util
  - 5.3.2.2.13.1. Class
  - 5.3.2.2.13.2. Contains useful methods to deal with IPv4
- 5.3.2.2.14. BroadcastConsummerBuffer
  - 5.3.2.2.14.1. Class
  - 5.3.2.2.14.2. A data structure used to hold information about a list of clients that are waiting in a queue
- 5.3.2.2.15. Parse_buf_log
  - 5.3.2.2.15.1. Class
  - 5.3.2.2.15.2. Processes a log file.
- 5.3.2.2.16. Refresh
  - 5.3.2.2.16.1. Class
  - 5.3.2.2.16.2. Used to reset network information
- 5.3.2.2.17. Linux_wlan_info
  - 5.3.2.2.17.1. Class

5.3.2.2.17.2. Used to acquire network information from a linux machine (Interfaces, SSID, and IP).

5.3.2.2.18. Mac_wlan_info

5.3.2.2.18.1. Class

5.3.2.2.18.2. Used to acquire network information from a Mac machine(Interfaces, SSID, and IP)

5.3.2.2.19. Win_wlan_info

5.3.2.2.19.1. Class

5.3.2.2.19.2. Used to acquire network information from a Windows machine(Interfaces, SSID and IP)

5.3.2.3. Design relationships

5.3.2.3.1. Linux_wlan_info, Mac_wlan_info, and Win_wlan_info shall provide information to BroadcastClient/Server

5.3.2.3.2. BroadcastData shall be used to store information for BroadcastClient/Server

5.3.2.3.3. InterfaceData shall be used to store information for Linux_wlan_info, Mac_wlan_info, and Win_wlan_info

5.3.2.3.4. BroadcastConsummerBuffer shall be used as a waiting queue for BroadcastServer

5.3.2.3.5. BroadcastClient’s output shall be used as BroadcastServer’s input.

5.3.2.3.6. Parse_buf_log shall retrieve information from files

5.3.2.4. Design overlay

5.3.2.4.1. Diagram provided below.

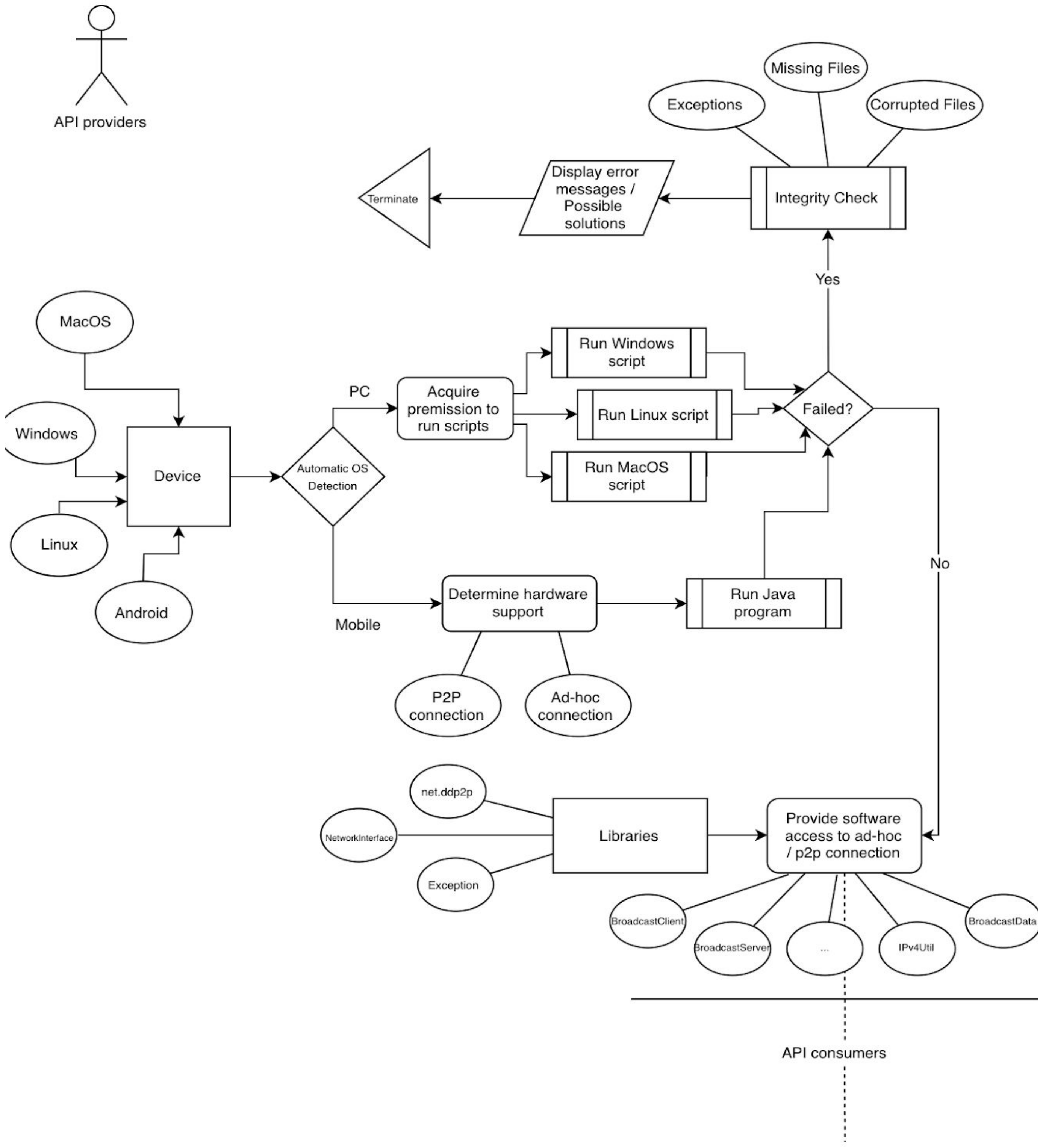

Fig. 1 (API providers viewpoint)

Appendices

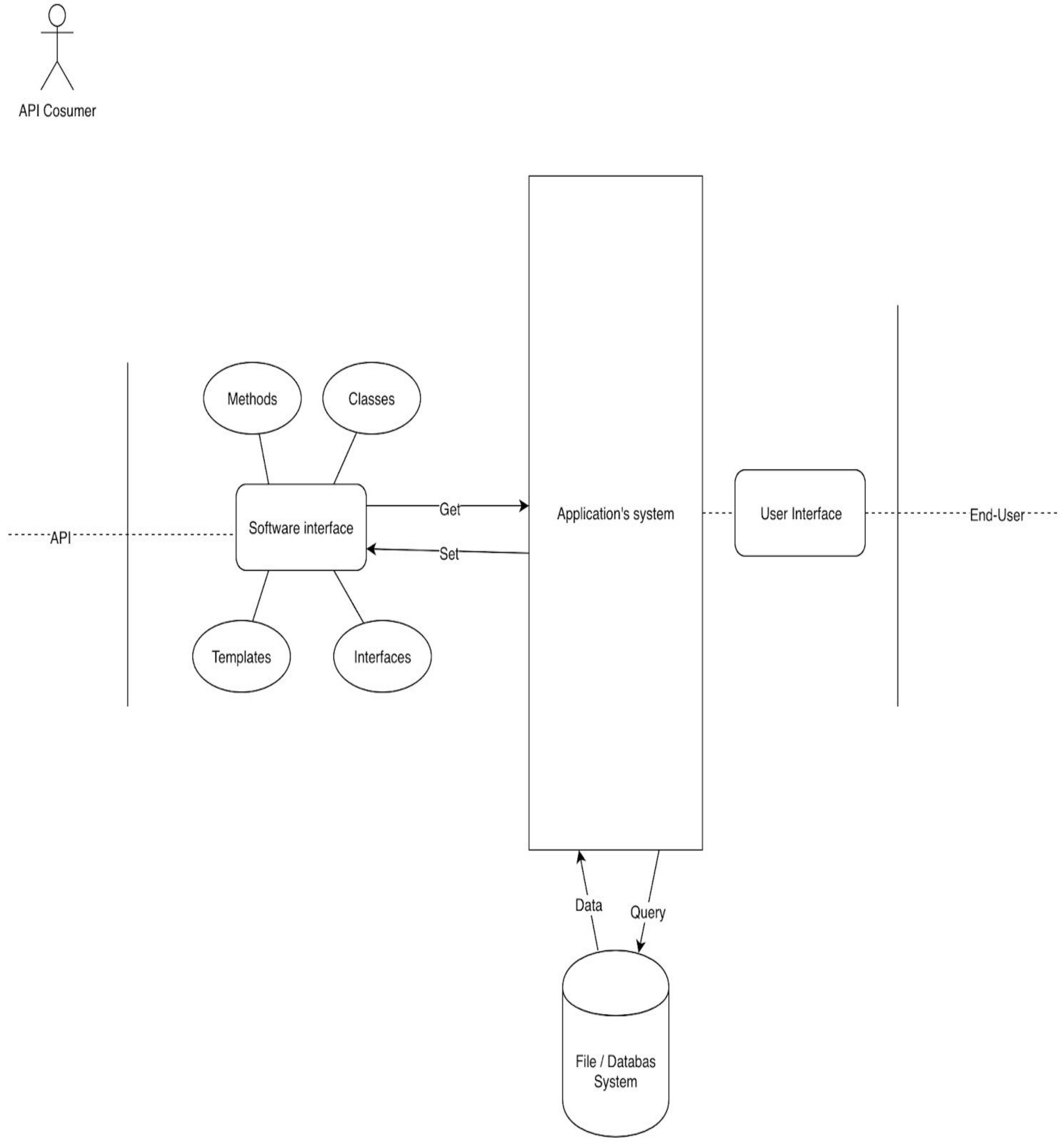

Fig. 2 (Diagram showing how API Consumers will work with the API)

Table of Contents