API Management Platform - Based on OutSystems

André Filipe Martins de Matos Santos
andre.matos.santos@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2019

Abstract

In recent years, application integration has shifted a lot. Disruptive technology is being developed at a great pace and Application Programming Interfaces (API) are, usually, in the middle serving as pillars. Either used internally, by partners or by the outside world, APIs are trendy and have been for quite a while now. However, and even though their ubiquity is undeniable, OutSystems does not have a way to expose the web services developed through the platform to neither usage while, on the other side, several API Management Platforms (AMP) allow the user to create, expose, monitor and monetize APIs. The goal of the project is then to take advantage of the Rapid Application Development (RAD) technology OutSystems provides on their platform and endow the capability to expose web services as APIs to its Integrated Development Environment (IDE) while also allowing to generate Software Development Kits (SDK) from any Swagger Specification. The report begins by setting the project goals. Then, the APIs' world is introduced, concretely by presenting the APIs themselves, the AMPs' most wanted characteristics and the existing tools in the market. After, the implementation process is explained, namely by detailing the system components and the possible flows of interaction with the solution. Subsequently the project results are shown and the conclusions are taken. The report finishes by enumerating the work to be done a few suggestions of my own on how to improve the solution.
Keywords: API; AMP; OutSystems; Swagger.

1. Introduction

The majority of the technology at people's hands, from the apps on their phones to the intelligent gadgets they interact with, are built using APIs as a basis. This, in a way, implies that on the consumers' side they should be easy to understand, test, use and evolve, while for API developers it is extremely important to display them in the most attractive and easy way possible in order to motivate their consumption. The OutSystems platform is optimized for RAD, and it permits developers to expose Representational State Transfer (REST) and Simple Object Access Protocol (SOAP) services in a few clicks, but not to publish them into an externally visible place.

1.1. Goals

Taking into account the needs of the API world, the goal of this project is to design, build and test an AMP that equips the OutSystems platform with a way to, quickly and easily, expose services as APIs. This aims to contribute to boosting both the APIs' exposure and consumption. The following functional requirements are expected to be fulfilled by the platform: (i) APIs exposed as reverse proxies; (ii) document APIs in the portal; (iii) test APIs directly in the platform; (iv) implement, at least, one policy of each type (i.e. security, mediation, service interaction and traffic management); (v) ability to download API SDK; (vi) generate API usage analytics report.

Regarding non-functional requirements, the solution must be: (i) intuitive; (ii) responsive; (iii) performative; (iv) maintainable.

The platform, which from now on I will refer to as "AMP", was developed using the OutSystems technology. It consists, basically, of a web portal to which API providers and consumers can access (i.e. users can be providers and consumers at the same time as there is no actual distinction). The portal list the publicly available API and, for each one, their methods, attributes and policies that can be applied. Even though it was initially aimed at the OutSystems developers, everyone who possesses a valid API definition in the JavaScript Object Notation (JSON) format is able to expose APIs at the AMP.

1.2. Existing solutions

There are plenty of solutions in the market and some are even open-source. However, when one decides to include an AMP in the business and starts studying the market, the results can be very overwhelming, especially because in general they all pro-

vide the same basic features. According to a study conducted by Forrester [1] which analyzed the "15 platforms that matter the most", there are a lot of strong contenders but there are only 5 which fit into the "leader" category: IBM API Management by IBM, WSO2 API Management by WSO2, webMethods API Cloud by Software AG, Rogue Wave Software by Akana and Apigee by Google, on which AMP is inspired. Figure 1, from Forrester's study report, shows the full list and where they stand against each other in the market.

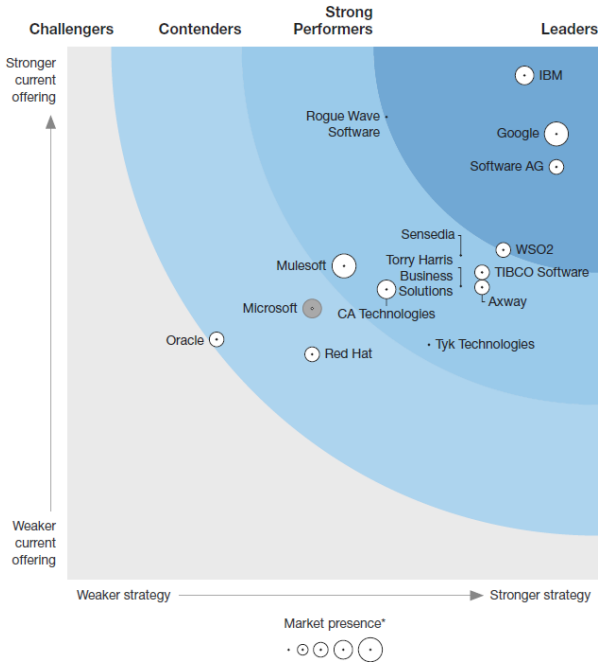

Figure 1: API management platforms value analysis

1.3. Report structure

The next section is where I introduce the tools and explain the concepts related to the subject. On the third section, I present the user flows and AMP's architecture in detail. On the fifth, I focus on showing the obtained results and comment on my expectations. The last section concerns the conclusion but I also take time to mention the work to be done and a few suggestions.

2. Related work

In this section, I dive into the state-of-the-art. Starting by introducing the APIs' world, I then demystify its concept against the one of service, with emphasis on both differences and similarities. Moving into the importance of APIs in our modern world, I will also tackle the value of the API management platforms and identify the trends. Moreover, I detail Swagger and the different tools it provides and finishes by giving an overview of the OutSystems platform.

2.1. Application Programming Interface

Given the generic designation of Interface in the informatics world, APIs can be defined as a shared contract between two entities with the intention of formalizing a common way for them to talk with each other and understand what is being said. So, since APIs are meant to communicate from system to system, it is correct to treat them as a set of protocols and definitions used to build software that, eventually, feeds on others applications information. In this context, I will relate to Web APIs [2], which are therefore a subset of APIs defined by a standard Hypertext Transfer Protocol (HTTP) request-reply message exchange typically formatted as JSON or YAML Aint Markup Language (YAML) [3]. YAML is a data serialization standard for all the programming languages, intended to be human-friendly, hence used to write API description files. As the request-reply process may be a simple one to understand, it might not be so easy to implement if the consumer is unaware about what kind of information the provider offers, how to request it and what to expect as a response. In short, APIs exist to make the implementation of such feature a not so complicated task for developers. For that to happen, they always come with a description of their capabilities, divided into three complementary parts:

- **Documentation:** it contains examples on how the developers can use its functions while being aware of the existing constraints. Being the responsible part for the API comprehension, it is divided in three layers [4]: (i) top-level; (ii) functional; (iii) technical.
- **Definition:** it is directly linked with the Documentation, mostly because the first can be generated from the latter. Yet, the Definition is meant to be understood by machines whilst the Documentation is to be human-consumable.
- **Specification:** it concerns about the API's general behavior as well as how it connects with the others. For this reason, it is most of the time confused with its Documentation, however, they are not the same thing. While Documentation describes with examples how the API is supposed to work (e.g. how to call their methods), the API Specification can be thought as the total explanation of the API, by combining elements of both the Documentation and the Definition. The Specification allows developers to check which functions are available, how to invoke them, what they will do, what kind of parameters are mandatory, what is the structure of the resulting reply, the type of objects that are received and so much more. To summarize, the Specification defines the supported data types which lead to the main design structure and consequently to the functional and expected behavior.

2.2. **API Management Platform**

AMPs are the linking channel between providers and consumers. They can be seen as a bundle of tools, features and services set together to allow managing and exposing APIs to the world. Making use of their compelling characteristics they respond to the needs of both user types. The following [5, 6, 7, 8], are a sum up of the most common and important ones:

- **Discoverability:** it is the first AMP benefit that tends to come up. As providers want to make their APIs public for use, consumers need a place to find what they are looking for. The solution is a common portal that impacts both user types.
- **Documentation:** this is, once again, a fundamental step for both user groups. For providers, exposing APIs with structured and appealing documentation (perhaps even generated automatically from the definition) is a must. On the consumer side, good documentation is how they can be sure if the API is what they are looking for or not.
- **Monetization:** overall an API is a mean to get a service done; and services are not, usually, for free. So, in the eventuality a consumer actually subscribes/invokes a company API, they need to pay for it. This is a process that differs a lot on how it is done [9] but overall the AMP role is to be able to identify who is making the API calls and inform the provider so that the billing can be done.
- **Analytics:** by monitoring the APIs behaviour, providers can better understand how much their APIs are used and, more importantly, how are the customers using them. Typically AMPs provide fairly interesting dashboards which are helpful in understanding what is really happening with the API (e.g. demographic stats, popularity rankings, performance measures, and more). From there, for instance, providers can better understand the customer needs, make business decisions and take the necessary actions.
- **Security:** besides "mandatory" is also one of the most flashy characteristics of this type of platform. It is important to notice that providers want to expose APIs, but they want it to be a safe thing to do. The APIs server should not be vulnerable to the exterior, so apart from exposing them through a reverse proxy (i.e. making APIs location agnostic), AMPs typically provide a set of other security measures from which the provider can choose in order to keep their content as safe as it can be. Consumer authentication, request limitations and validations or IP filtering are some of the common key cases platforms make available for customers.

2.2.1. **Apigee**

Apigee, which is property of Google since 2016, provides the leading API platform (i.e. Apigee Edge) for enterprises and developers to design, secure, scale, analyze and monetize APIs. In short, when using the platform, users are able to manage the entire API lifecycle and with Google being one of the co-founders of the Open API Initiative (see section 2.3), Apigee delivers a powerful platform completely integrated with Swagger tools.

The platform focus on two key aspects: people and technology. Their goal regarding people is to connect the application developer (consumer) with the API team (provider) and in order to do this, Apigee provides analytics and developer channel services. Regarding technology, their aim is to connect the APIs with the apps by providing gateway and app services.

With a focus mainly on app developers, the platform disposes of a developer portal that allows API providers to show their work to whomever is interested, in a clean and appealing way. On the other side, these APIs are exposed through gateway services responsible to ensure safety and mediation between the applications and the backend servers. In fact, the exposure of the API through the proxy allows it to be customized with different available policies [10] which, according to Apigee itself, enable to "augment your API with sophisticated features to control traffic, enhance performance, enforce security, and increase the utility of your APIs, without requiring you to write any code or to modify any backend services". These same proxies are also monitored allowing both API consumers and providers to evaluate the API behaviors depending on the circumstances — e.g. as it might be useful for an app developer to monitor both API and app behavior (developer channel services), for an API provider what matters is the business information that is extracted from the API usage (analytics services).

As mentioned in section 1.2, AMP was built with great inspiration on Apigee. In the previous paragraph I noted a few strengths of the platform which, in fact, match the functional requirements set in section 1.1, namely: (i) APIs exposed in gateways services (i.e. proxies); (ii) developer portal where providers show their work (i.e. documentation and testing); (iii) proxy customization (i.e. policies); (iv) proxy monitoring and API behavior evaluation (i.e. analytics reports). As a matter of fact, the only AMP functional requirement that Apigee does not fulfill is the possibility to generate an SDK for any given API.

2.3. **Swagger**

Swagger tools are one of the most important pieces of API development nowadays. In a way, it started in November 2015 when Swagger announced a partnership with the OpenAPI Initiative (OAI) [11]. On

the last day of the year, they donated Swagger Specification which changed its name and became known as OpenAPI Specification (OAS).
OAI is an agglomeration of industry expert companies [12] that envision the value of standardizing the way REST APIs were described, more specifically with a goal based on creating a standard for a language-agnostic interface directed to RESTfull APIs. This standard is intended to be a way for both computers and people to be able to analyze and understand a service without having to look neither into its source code nor to its documentation. Summing up, the core idea is that a consumer should be able to easily comprehend and interact with the service with low effort and only a tiny amount of implementation logic.
Then, an OpenAPI definition document can be used to achieve different goals such as displaying the API by using documentation generation tools — interactive documentation —, generating code in several programming languages for both servers and clients, testing and more.
Despite the changing of name, the different existing tools remain known as Swagger due to a number of reasons, the most important being the recognition and contribution from the existent community at the time. Since the start of the partnership those tools evolved, got a huge boost, and are now the pillars beneath a great percentage of most used software in the world.
In the following sections, I will introduce and briefly explain the different tools Swagger provides, referring to their purpose and importance.

2.3.1. **Swagger Editor**

This is the tool that allows writing APIs' specifications [13, 14]. Either designing from scratch or editing existing ones (i.e. it is possible to import from either a file or a URL), this editor provides a set of useful functionalities to the developer like automatically rendering the content of the specification and allowing to interact with it while still defining. It is available to download and install for any environment but can also be used in its cloud-based version.

2.3.2. **Swagger UI**

This tool, just like Swagger Editor, generates a visual representation of the API specification while also allowing to perform calls to said functions, verify inputs and outputs, documentation and more [15, 16]. It can be seen as part of the Swagger Editor functionality since the editing part is not available for this tool. However, this is intended for API consumers that might be interested in checking the API behavior without the need to have the logic implemented.

2.3.3. **Swagger Codegen**

This tool is able to generate server stubs and client libraries in most popular programming languages directly from an API specification [17, 18, 19]. As a result, Swagger Codegen allows to improve the API consumption numbers, and both ease and speed of developing an application on top of it.

2.3.4. **Swagger Inspector**

This tool, once again cloud-based, allows to perform requests to web-services exposed using REST, GraphQL or SOAP [20, 21]. Despite allowing to invoke a specific endpoint of an API, Swagger Inspector offers the possibility of using its definition by making a call to the root URL. This will import all the available functions, making easy to select which one to test.
Swagger claims that this is the best way to create OAS from the existing APIs, and they actually facilitate it by storing the history of the functions' invocation. After, it is possible to select the ones desired from the list and generate new documentation with those chosen endpoints.
Swagger Inspector is the last released Swagger tool (i.e. it was not existing at the time the AMP development started) and it closed a big gap on API testing by making able to do it directly from any browser.

2.4. **OutSystems**

OutSystems is a Portuguese company born in 2001 and is the main contributor to the development of this project. As implied in the title, the AMP was developed using the OutSystems tools and technology. The company provides a cloud solution for application development in the form of Platform as a Service (PaaS) which is, at the moment, considered the number one low-code platform for RAD in both web and mobile worlds [22, 23, 24, 25]. OutSystems uses a Continuous Deployment (CD) model, allowing the developer to use their 1-click button to immediately publish the applications [26].

2.4.1. **Service Studio**

Service Studio, their IDE, allows developers to build complex applications either to mobile or web platforms by leading their focus into four main categories, which are presented in different tabs:
- **Data** tab is dedicated to the creation and management of entities (i.e. keyword OutSystems uses to refer to database tables), structures, session variables and site properties.
- **Logic** tab is dedicated to the creation of server actions and integrations (i.e. consume and expose REST and SOAP services) and manage roles.
- **Interface** tab is dedicated to the UI design and how the different pages connect with each other. To note that before each screen load, a special ac-

tion, known as "preparation", is executed in order to apply some logic (e.g. access permissions checks) and to fetch every needed information from the database.
- **Processes** tab is responsible to define the workflow behind an application. Typically this is used when a user needs to interact with other (or even itself), but depend on other tasks (i.e. asynchronism). In short, it is used to define processes and timers.

2.4.2. Integration Studio

This is an IDE meant for proficient software developers to write and deploy custom extensions, which can be actions or entities. Action can coded in Java, .NET or both platforms. An entity extension is basically meant to be a connection to an external database. Such extensions can after be integrated into the application through the integration folder in the logic tab of Service Studio.

2.4.3. Forge

OutSystems Forge [27] is a community repository. Developers typically contribute with open-source projects, such as widgets, themes or even full applications. which can then be reused to speed up the development process.

2.5. Problem statement

Even though OutSystems's IDE is ranked the best one in the market, it still lacks some features. Users of the OutSystems tools are able to deliver their products faster but, at this stage, an easy and quick way to expose those services as APIs in an AMP is missing. A simple, yet optimal, way to deal with this gap would be if either the IDE or the service documentation page possess a mechanism (e.g. a button) to directly publish an API on some AMP.

2.6. Objectives and contributions

Taking into consideration the suggestion in section 2.5, the objective of this dissertation is to use the OutSystems platform and technology to build a simplified version of an AMP powered with the fundamental features described in section 2.2, just like any other. Due to the obvious impossibility of altering both Service Studio and the documentation page to include the mentioned mechanism, the result of this project is not to be integrated in the OutSystems platform, even though the used technology is their property.

3. Implementation

In the context of this section, I will introduce the system components and the detailed user flows: (i) of the provider when exposing an API; (ii) of the consumer, when browsing, testing and downloading the different APIs; (iii) of the final user, when invoking APIs through an application.

3.1. System components

- **Portal:** the entry point on AMP. Shared amongst providers and consumers, allows users to expose new APIs, look for already published ones, test them and generate an SDK library out of their definition. Moreover, it provides graphical analysis of the API consumption, however such functionality is exclusively for providers.
- **Proxy:** component responsible for directly dealing with the API but serves other purposes: (i) mask the real API address; (ii) filter the requests and secure the access to the API; (iii) transform the message content; (iv) abstract the services complexity contributing to the API lifecycle management. In short, this component's goal is to receive requests from consumers, apply eventual policies, forward the request to the API and return the response.
- **Database:** stores not only the API definition content but also the API information (e.g. service functions, response types, attributes and more), the user subscriptions, the proxy logs and associated policies.
- **Codegen:** exterior to the platform (i.e. it is a Swagger tool), is available through an API and is responsible for allowing users to retrieve the APIs' SDKs.

3.2. Provider flow - Publish

This is the flow describing how a user exposes an API in AMP and it relates to Fig. 2. At first, the provider must access the portal and open the "New API" popup. Publishing an API is only available for signed-in users, so they must be authenticated as a pre-condition. When the popup shows, the provider should (i) name the API, (ii) write a brief description, (iii) paste the corresponding URL, (iv) choose, from a list, the available policies to be applied and finally (v) choose if the API will be immediately visible to the public. Then, AMP will fetch the API definition from the provided URL, parse it and store the API information in the database. Lastly, AMP builds a proxy for the API and redirects the user to its specification page.

3.3. Consumer flow - Search, test & download

This is the flow describing how a user searches for information about an API (Fig. 3), tests (Fig. 4) it and eventually downloads its SDK (Fig. 5).
At first, the consumer must access the portal an open the "All APIs" page. All the publicly available APIs are listed and, for each one, the user is able to see its name and summary, the provider's name and email, and its methods and available policies. Users also have a search tool available which they can use to search by the name and summary of the API they wish to find.
By clicking in the API name, the user will be redirected for its "specification" page where all the API

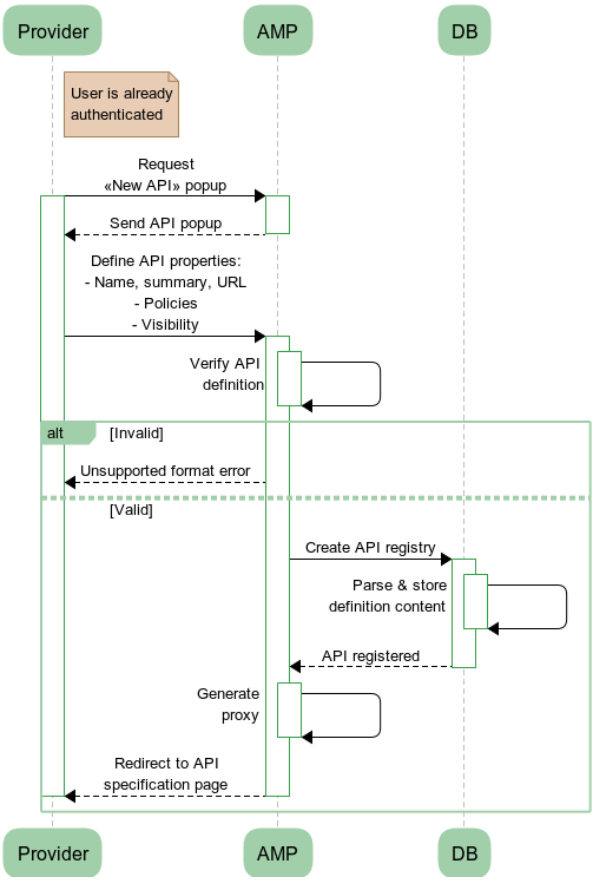

Figure 2: Sequence diagram for publishing an API

information is available. In there, users can look into the API definition (i.e. a URL on the top of the page will open a new tab with the content, if clicked) and for each API method the input and output parameters (with examples), the possible responses and the request URL.
Also, the page contains a "Try out" button which enable users to test the API methods right from the portal if they choose to do so. If they do, an HTTP request is sent to the API proxy which in turn will maps to the correct API address, apply any policies that may be needed and forward the request to the API server. After receiving the response, more policies might need to be applied and then the final result is shown on the page.
Apart from testing, there is also the possibility to download the API SDK, yet, only signed-in and subscribed users are allowed to do this action — API owners are automatically subscribed. When the "Generate library" button is clicked a popup is shown prompting the user to select the desired type (i.e. client or server) and the programming language. After choosing and confirming, AMP requests Swagger Codegen for the corresponding stub and then downloads it into the user device.

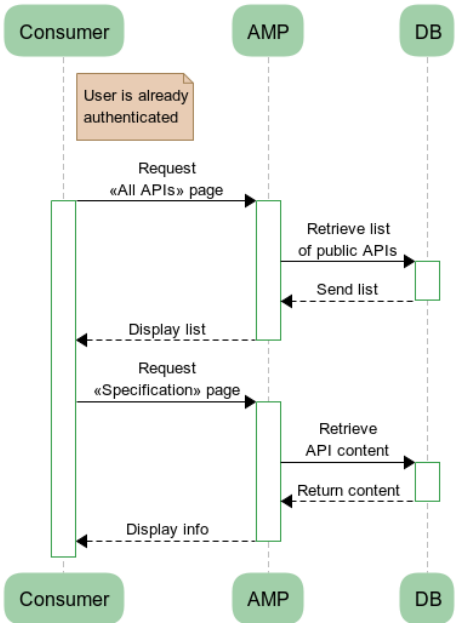

Figure 3: Sequence diagram for API investigation

3.4. **Final user flow - Invoke**

This is the flow describing how an API invocation is done on AMP. For the final user, this process is a black-box as they are triggering APIs by interacting with an application. That said, when the proxy first receives the request from the app, it tries to find an API key in the request header. If it finds and is a valid one, the request is mapped to the correct URL and forwarded to the API server. After receiving the response, the proxy sends it back to the app so the results are displayed to the user. Just like the "test API" situation, if any policies are required to be executed in the proxy, they will after the request is received or before the response is sent. This time I am not providing a sequence diagram since the flow is very similar to the one detailed in Fig. 4 — differences are the initiator of the action is the final user "App" and not the AMP, and there is an API key check before mapping the URL, which returns an error if it is not found or not valid.

4. Results

This chapter describes an overall representation of the developed work, more specifically the concept results. For each functional requirement proposed in subsection 1.1, I will include an image and briefly explain how I was able to match it:
- For requirement "APIs exposed as reverse proxies", as a matter of fact, I exposed an API which accepts GET, POST, DELETE and PUT requests. The requests receive a Global Unique Identifier (GUID) as a parameter which is then used to identify the correct API. For GET and DELETE requests, it also expects an array if any values are

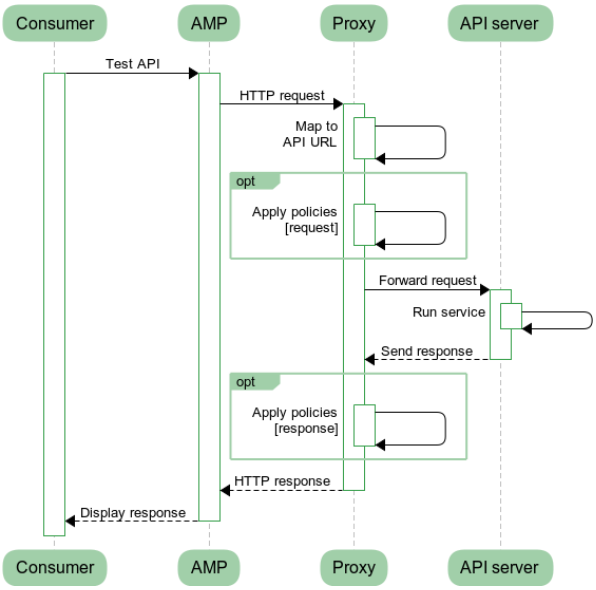

Figure 4: Sequence diagram for testing an API

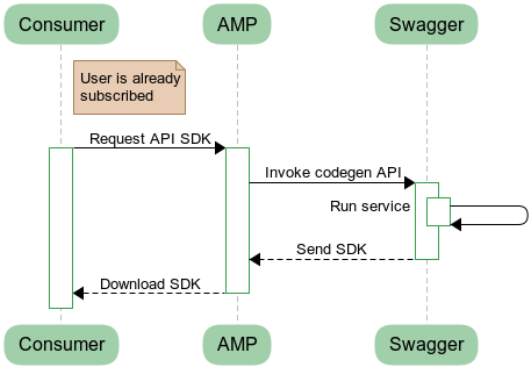

Figure 5: Sequence diagram for SDK download

needed to be passed along to the true API. When the request is received, the correct URL is fetched from the database and the attributes are mapped. It is at this time that the policies are applied. Then, the final URL is built and the API server is invoked. Fig. 6 mimics the functioning of the proxy.

- For requirement "Document APIs in the portal" to be possible, all the APIs' content is parsed and stored in the database at publication time. Then, the "API specification" page is made available to the users. A preview can be seen in Fig. 7.

- The "Test APIs directly in the platform" requirement is possible due to the combination of the "API specification" page and the proxy. The test mode is activated from inside the page and when the test is triggered, the request is sent to the proxy and the flow which was previously explained will be executed. Fig. 8 shows the result of a test being shown to the user, in the platform.

- As for the policies, I ended up not implementing any policy regarding "traffic management" yet I included two on the "mediation", two on the "security" and one of the "service interaction" groups. "JSON to XML" and "XML to JSON" are the mediation ones and always options for both provider and consumer sides. In respect to security, I implemented the API key concept. The key should be passed in the header of every request as it will be analyzed by the proxy. If the key is invalid, an error message will be thrown, otherwise the proxy will call the API server. The second security policy which I implemented was the OAuth2. As for the authentication, the only Trusted Third Party (TTP) implemented at this point in time is Google. A "Sign in with Google" button is available at the "Sign in" page, as shown in Fig. 9. The button, when clicked, displays the TTP authentication popup from where the user can authenticate himself — Fig. 10. In order to experience this capability, users must possess Google accounts. In case they do not, there is still the possibility to register the old fashion way: with an email and a password.

- The "Ability to download API SDK" requirement is possible thanks to the Swagger Codegen tool. Fig. 11 shows the popup displayed to the user for him to choose to which framework the SDK is needed.

- The last requirement "Generate API usage analytics report" is possible due to the "Message logging" policy. In the "My APIs" page there is the "Usage Report" button. When clicked the platform displays a popup showing two graphs: (i) column chart which shows how many API calls happened, per day, during the past five days and (ii) a circular chart detailing the frequency of the invoked methods. An example of this popup can be seen in Fig. 12.

In the same popup there is a "Download report" button which, when clicked, will download a .xlsx file containing the details of the proxy activity, as shown in Fig. 13.

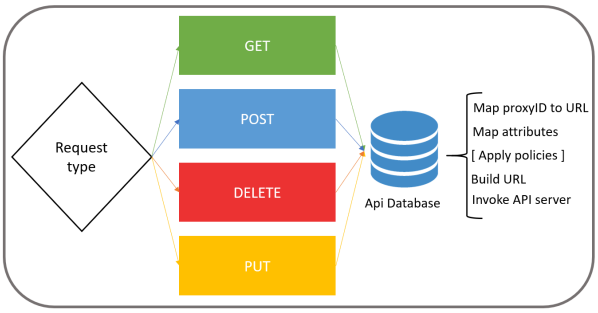

Figure 6: Inside of the proxy.

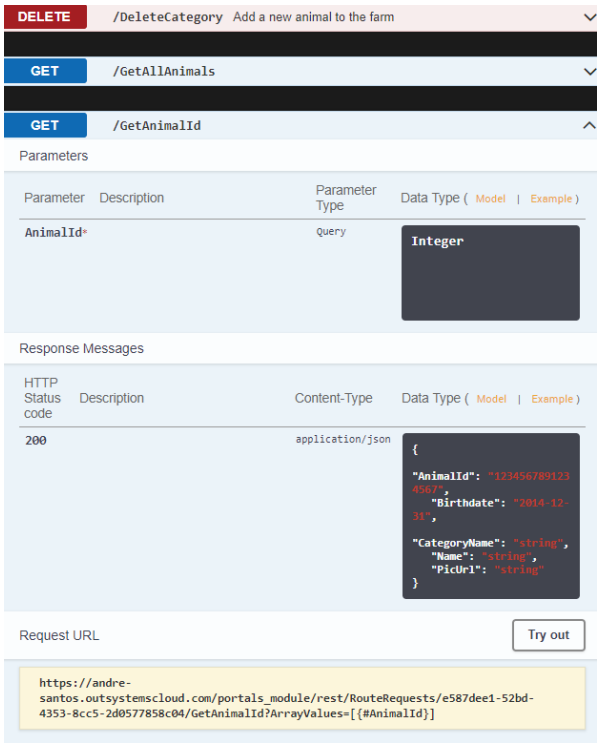

Figure 7: API method examination.

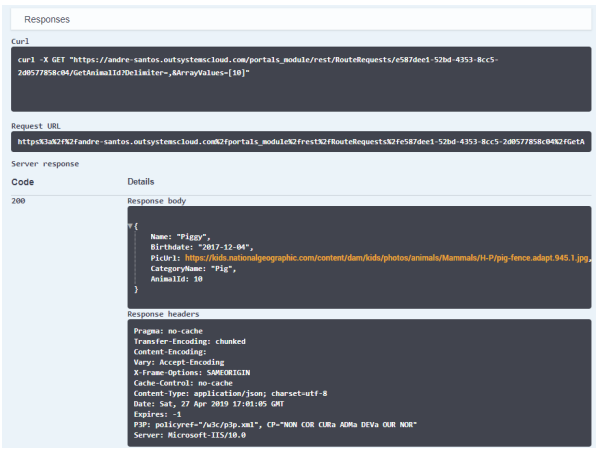

Figure 8: Test results on the platform.

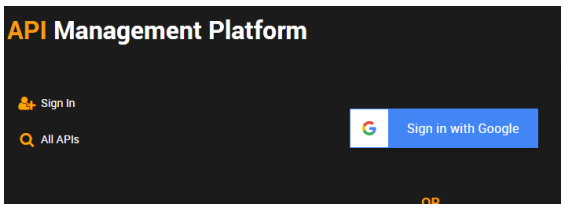

Figure 9: Sign in with Google button.

5. Conclusions

The market of API management platforms is already partially saturated, as dozens of vendors are providing their solutions. Reports [1, 28, 29] show the market currently is lead by five products, but more are very close to be considered equals. As they all focus on some selling point such as analytics, monetization, security or even the pricing plan, no platform focus on a specific environment and that is when AMP comes in. No known platform is build based on OutSystems which by itself is a differentiator between this platform and the others. Even though AMP should be considered as a prototype, after the provided evaluation it is observable that all the basic functionalities were implemented and are working, with exception for the monetization.

5.1. Future work

This last part of the report entails features which were part of the initial scope but were not implemented and a few suggestions that could help to evolve the platform closer to an Apigee look-alike.

5.1.1. Not implemented features

- **Multiple TTPs:** So far the platform only supports one authentication provider: Google (i.e OAuth). However, it is designed to allow the users to benefit from all the ones whose credentials are in the database, more concretely in the "TrustedThirdParty" table. Whenever the credentials of a new one are introduced in the database, the customer will be able to use it the next time an authentication needs to be performed.

- **OpenID:** This policy is categorized under the security group. It allows users to authenticate towards the API manager without introducing their sign in credentials directly in there. In practice, the users obtain an identity confirmation token from an identity provider (e.g. Google), which is then routed to the relying party (i.e. AMP) to be checked. The policy will then authenticate the user if the token was provided by a TTP.

5.1.2. Suggestions

- **Application concept & app key:** While in AMP the concept of "user" exists, the notion of application does not. With that in mind, as the platform already has the API key policy implemented (section 4), it lacks an App key policy. Even though this is something that could only be implemented after the platform recognizes what an application is, an app key is a nice-to-have feature as it allows to identify and bill the actual consumer of an API. As of now, AMP allows an API to be called from a various number of applications with a single key but with the combination API key + app key it would be able to recognize which applications are allowed to call which APIs while also identifying if the application owner is legit.

- **More monitorization:** Key factor for any API

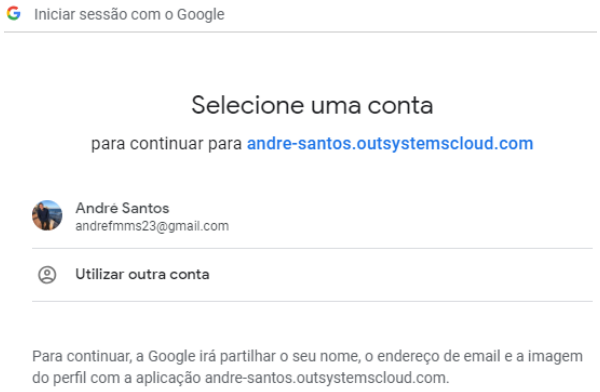

Figure 10: OAuth popup.

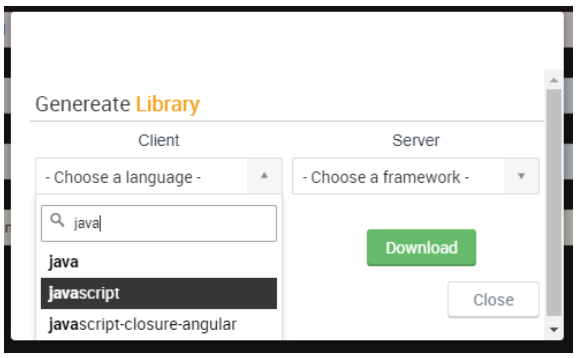

Figure 11: Popup showing possible API's SDKs.

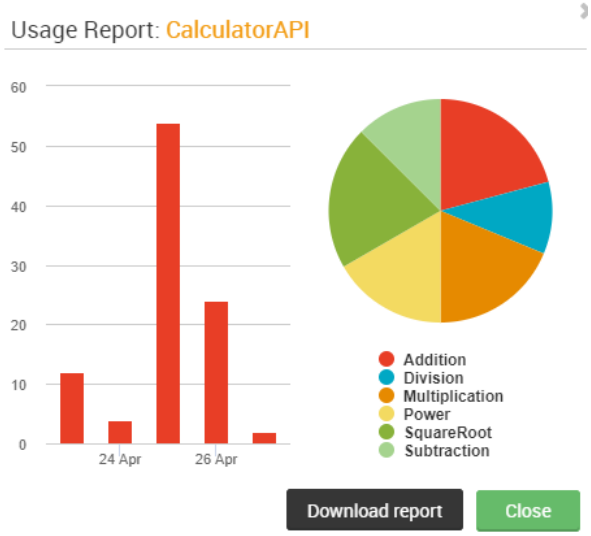

Figure 12: Charts with API proxy activity levels.

	A	B	C	D	E	F	G	H
1	Data	FlowTy	MethodNar	RequestTy	TimeOfDa	Date	Email	IsTest
2	[66,21]	REQ	Addition	GET	10:32:19	2019-04-25		VERDADEIRO
3	87.0	RES	Addition	GET	10:32:20	2019-04-25		VERDADEIRO
4	[555,36.5	REQ	Multiplication	GET	13:19:50	2019-04-27	andrefmms23(	FALSO
5	20257.5	RES	Multiplication	GET	13:19:50	2019-04-27	andrefmms23(	FALSO
6	[3{char}:	REQ	Power	GET	15:09:16	2019-04-24		VERDADEIRO
7	{"Errors":	RES	Power	GET	15:09:17	2019-04-24		VERDADEIRO
8	[3051]	REQ	Power	GET	15:09:56	2019-04-24		VERDADEIRO
9	{"Operati	RES	Power	GET	15:09:56	2019-04-24		VERDADEIRO
10	[0.1,0.1]	REQ	Addition	GET	19:59:54	2019-04-25		VERDADEIRO
11	0.2	RES	Addition	GET	19:59:54	2019-04-25		VERDADEIRO
12	[2,44]	REQ	Subtraction	GET	20:01:29	2019-04-25	andrefmms23(	FALSO

Figure 13: .xlsx file containing the details of the proxy activity.

management platform, monitorization tools grants API providers and AMP administrators a graphical overview, at different scales, on several kinds of statistics and results related to the API usage. As an example, providers can see which of their APIs is more popular, subscribed and/or downloaded, while administrators can understand the policy usage rate or, for instance, if any policy is causing a big increase on some response time. The message logging policy is a starting point for the implementation of this feature, as the charts for API usage can be generated from the existing logs data and, as the first evolves the latest can too.

- **AMP Monetization:** Crucial for every business, monetization is also a great deal with API management platforms, especially because it can be done in quite a few different ways. Apigee, for instance, provides billing plans which consist of different bundles made of: (i) number of allowed users, (ii) number of authorized API calls per month, (iii) number of days included in the analytic reports, (iv) support type and (v) SLA. Considering the final result of AMP, a good commencement would be to also charge providers based on personalizable unit bundles composed of: (i) a number of exposed APIs, (ii) a number of allowed calls per API and (iii) a limited number of policies by API. In concrete, users would pay only for the number of APIs they expose, with the limit of calls they choose and the number of policies they want to benefit from.

- **API Monetization:** Getting paid per finished transaction is the typical way API management platforms operate (e.g. that is Apigee's business model [9]). As a new platform, AMP should differentiate itself and charge customers a fixed price based on their API popularity. In practice, this idea allows customers to pay an amount per transaction that fluctuates depending on the number of calls an API gets. With the continuous evolution of the platform and the addition of new policies, the business model can be adapted to discern premium policies from not-so-special ones and possibly charge different values for them.

- **More policies:** The more relevant policies, the better. Some can be quite challenging to develop, in particular if they can be configured with different values for different API proxies. Being that AMP does not have any traffic management policy available, these are the two I find more important and would like to have implemented:

- **Quota policy** is especially interesting because it allows to define the number of request messages that an API proxy accepts over a period of time. Ideally, the user should be able to set a counter value to the number of allowed API calls and, a time measure (i.e. second, minute, hours or day) and a time amount. Then, the policy will let through the requests and decrease the counter by one while it is not equal to zero. After the predefined time, the counter will be reset to the initial value.
- **Spike arrest policy** enables the proxy to play defense in case of an attack. The user should simply set a number of authorized requests to per time unit (just like the quota policy). On this case, the policy should work by throttling the time passed from the last incoming message and not by a counter.

Even though these two policies may seem similar in the first place, they serve two purposes. Quota policy exists to limit the number of connections the consumers can make to the backend during a specified time interval. Spike arrest is idealized to protect the proxy again a spike in the requests or a Denial of Service (DoS) attack. As an example, a proxy can be configured to accept 1000 requests per minute with the quota policy but also to reject more than 5 per second with the spike arrest policy meaning they can work simultaneously.

References

[1] Randy Heffner. The Forrester Wave: API Management Solutions, Q4 2018, 2018.

[2] Werner Vogels - Amazon and the Lean Cloud on Vimeo.

[3] The Official YAML Web Site.

[4] What is the Difference Between API Documentation, Specification, and Definition? - Nordic APIs.

[5] API Management - Features.

[6] API Management Platforms Capabilities.

[7] The Definitive Guide to API Management © CC BY-SA. Technical report.

[8] Top 10 Best API Management Tools with Feature Comparison.

[9] Monetization overview - Apigee Docs.

[10] Policy reference overview - Apigee Docs.

[11] Introducing the Open API Initiative!

[12] Current Members - OpenAPI Initiative.

[13] Swagger Editor - API Development Tools.

[14] Swagger Editor - Documentation.

[15] Swagger UI - API Development Tools.

[16] Swagger UI - Documentation.

[17] Swagger Codegen - API Development Tool.

[18] Swagger Codegen # Online generators.

[19] Swagger Codegen - Documentation.

[20] How to Use Swagger Inspector.

[21] Swagger Inspector - API Testing & Documentation Tool.

[22] Clay Richardson and John R Rymer. The Forrester Wave: Low-Code Development Platforms, Q2 2016 - The 14 Providers That Matter Most And How They Stack Up. Technical report, 2016.

[23] Jeffrey S Hammond. The Forrester Wave: Mobile Low-Code Development Platforms, Q1 2017. Technical report, 2017.

[24] Gartner Peer Insights. Gartner Magic Quadrant for High-Productivity Application Platform as a Service (hpaPaaS) 2017. Technical report, 2017.

[25] OutSystems. The State of Application Development. Technical report, 2017.

[26] OutSystems 2-Minute Overview.

[27] Forge FAQ.

[28] Magic Quadrant for Full Life Cycle API Management.

[29] Rob Koplowitz and John R Rymer. The Forrester Wave: Low-Code Development Platforms For AD&D Professionals, Q1 2019 - The 13 Providers That Matter Most And How They Stack Up, 2019.