REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE IBN KHALDOUN - TIARET

MEMOIRE

Présenté à :

FACULTÉ MATHEMATIQUES ET INFORMATIQUE

DÉPARTEMENT D'INFORMATIQUE

Pour l'obtention du diplôme de :

MASTER

Specialty: Genie Logiciel

Par:
Toumi Mustapha Abderrahmane

Sur le thème

Integration des services métiers des entreprises via un workflow flexible

Publicly Defended on 18 / 07 / 2021 in Tiaret in front of the jury composed of :

Dr CHIKHAOUI Ahmed		University Of Tiaret	President
Dr. OUARED Abdelkader		University Of Tiaret	Supervisor
Dr TALBI Omar		University Of Tiaret	Examiner

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

ACKNOWLEDGMENTS

At the outset, i'd want to thank God for enabling us to accomplish this humble job, which was done with God's aid, then, with the help of those who assisted us, and finally, with our own preparation and labor, from those who supported me and contributed to this work, I'd like to begin with Mr. Taher Ouhrouch, the owner of stockAndBuy, the one who provided us with a wonderful opportunity to work with him, and i learned a lot from him, he provided us with a lot of very essential information, he was a great cause for me to learn new and vital things, i have benefited greatly from his wealth of knowledge, i want to say thank you to Mr. Taher Ouhrouch because he gave us the chance to learn from pluralsight which one of the best e-learning platforms, and i want to express my appreciation and gratitude to my Supervisor Dr. Ouared Abdelkader, he was a tremendous assistance to us, he guided us through this trip in an outstanding manner to say the least, i am extremely grateful that he took me on as a student and continued to have faith in me, his advice also lays between the lines of this theses.

Also, my colleague Ali Kouriba, who was a fantastic person during this period and since I met him, and Mr. Fethi Boukhors, who was a good guy and deserves a lot of credit for introducing me to two individuals from whom I learnt a lot, i already mentioned Mr. Taher Ouhrouch This project would not have been possible without him.

Then i can to move on to Dr. Mourad Bouach, who has always encouraged me and told us that we have the power to change even if we're still at the beginning or our circumstances are bad and inappropriate, and that we should never lose faith in ourselves; he had a very positive impact on us.

And i want to thank Mr. Zakaria Mansouri and Dzcode community this theses built based on what i learned by contributing to the amazing open source project of this community.

I want to say thanks to my parents, i like to thank my dear ones who did not spare anything but handed them to me, immense gratitude as always to all of those who supported me.

ABSTRACT

Web APIs Lifecycle has become a driver for business in various domains (e.g. CRM, marketing platforms, etc.), hence, the way in which organizations use of Web Services to introduce the various functionalities for their E-business systems using API components (e.g. Magento, Shopify, Paypal). TheWeb APIs Liefcycle process requires a deep knowledge related to many aspects: RESTful API, DevOps, Agile methodologies, framework and tools such as Unit testing factors of basic functionality, etc, currently, the development of Web APIs is still time-expensive due to the time allowed for developing and integrating in order to obtain API. close to the practical usages of Business Process.

Moreover, an API can concern simultaneously many resources, hence its development may require it to be done on a collaborative way between actors who are consumers or customers, API manager, software architect, developers who complement each-other, since each one of them can be an expert in a particular resource domain, In this theses, we propose a framework which helps and assist API stakeholders in the phases of the API lifecycle, to work collaboratively and enables them to automatize and ease the traditional workflow dedicated to carry on the workoflow an API lifecycle through E-business Process.

This framework is implemented as model-based infrastructure thanks to model-driven engineering settings, in contrast, also we provide a prototype of a system to interact with the API Lifecycle process.

CHAPTER 1. MOTIVATION AND RESEARCH METHODOLOGY

1.1 Motivation

Modern web APIs were theoretically born on 2000 with [18], for the industrial sector, Salesfoce may have been one of the first enterprises to recognize the value of data-driven change on February 7, 2000 according to [31], after that Amazon launched the Amazon Simple Storage (S3) and Amazon Elastic Compute (EC2), the first offers a basic storage service, and the second one provide servers that developers could leverage to deploy the necessary infrastructure for applications both were web APIs, this was a rallying point for transferring everything to the cloud, as a result, APIs paved the way for a new age that will dominate the coming decades, it also aided a number of businesses in solving challenges and expanding their business.

1.2 Problem Statement

In addition to this, we saw that web APIs became so important it changed the way people do business on the web, and it also resulted in the appearance of new fields, it has become essential to provide multiple stages that a team or a one developer must follow whenever creating an API, and this has become a separate field, with the name of these phases together being API Lifecycle.

But each organization, developer, or enterprise has their own view of API Lifecycle, also, some treat API as software, but API should be treated as a product [24], which was a problem for junior developers and those with little experience in the field of developing APIs, most of them have no idea what they should do for the creation of the API or what the phases of the web API Lifecycle are, and this is a big issue that we experienced while working on the StockAndBuy web API, which led us to be late in finishing the job and not as anticipated.

Our goal in this thesis was to begin working on a unified API Lifecycle view and to create a framework to help developers and small teams in their journey of API creation.

1.3 Research Questions

In the next part, we offer a two research questions, which comes from the above noted issue statement and will lead this thesis throughout the entire study:

Reasearch Question 01: How can we provide a Framework and a set of successive phases and guidelines to demonstrate the path of an API Lifecycle for a small team or a developer?

Our aim was to create a framework called *COVERING-API* which is based on its core components and the different links between the different stages in a given project to create a complete and a usefull API product that the consumer can benefit from its usage.

Another goal, when we asked this question, which has motivated us we have proposed a design languages for *COVERING-API* as an intermediate framework between API developers and API product manager, this framework is based on the *COVERING-API* language which is conceived as a DSL(domain-specific language) for *COVERING-API* management system.

We also attempted to begin an effort to establish a unified perspective of the API Lifecycle with this question.

Reasearch Question 02: How can we implement this framework as a tool to help a small team or developer create a successful web API product?

In order to answer this question, we planed, designed, and then implemented a web application prototype, to help the developer or small team, and guide him as well as provide a guidance about what he should do in this stages, in addition to helping him and suggesting the best thing to do as a next step, depending on the *COVERING-API* framework.

This tool should be able to analyze the main repository in which the team works, and encouraging the user to follow metrics and guidelines in order to ensure better collaboration for

the team, and more inclusion for one developer, it should provide a simple way for the team leader to follow what other members are doing during the course of the project.

1.4 Our Research Approach

Our research approach in this theses is based on the design science framework [54], and [27], [54] this study offers a new paradigm that incorporates activities such as (1) theory building, (2) solution technology invention, (3) artificial evaluation, and (4) naturalistic evaluation check figure 1.1 for more details.

We followed these stages and imposed hypothesis first, then we started with the design science process, and then we carried out the naturalistic evaluation stage in the form of a survey study, knowing that this is just one iteration and several iteration must be done in order to complete the work in an integrated manner.

As John Venable said that invention of solution technology lies at the heart of Design Science Research, a solution technology is any strategy to improving an organization, such as information systems, information technology, systems development methodologies, algorithms, management practices, and a variety of other technologies or approaches.

The invention of solution technology entails the high-level and thorough design, construction, and potentially functional testing of a hypothesised solution technology.

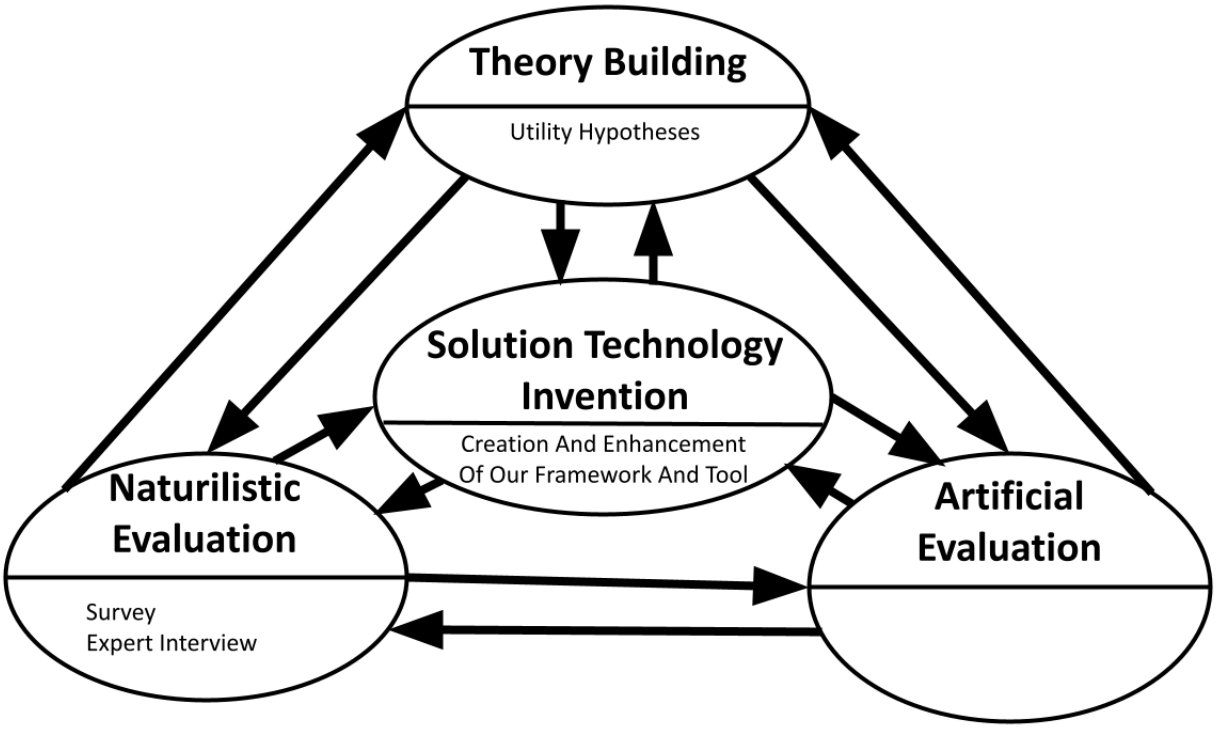

Figure 1.1 Framework and Context For Design Research From [54]

In our application of design research in an information system we relied on [27], who says that research in this area relies on three main parts which are the environment, the information systems (IS) research and the knowledge base in figure 1.2.

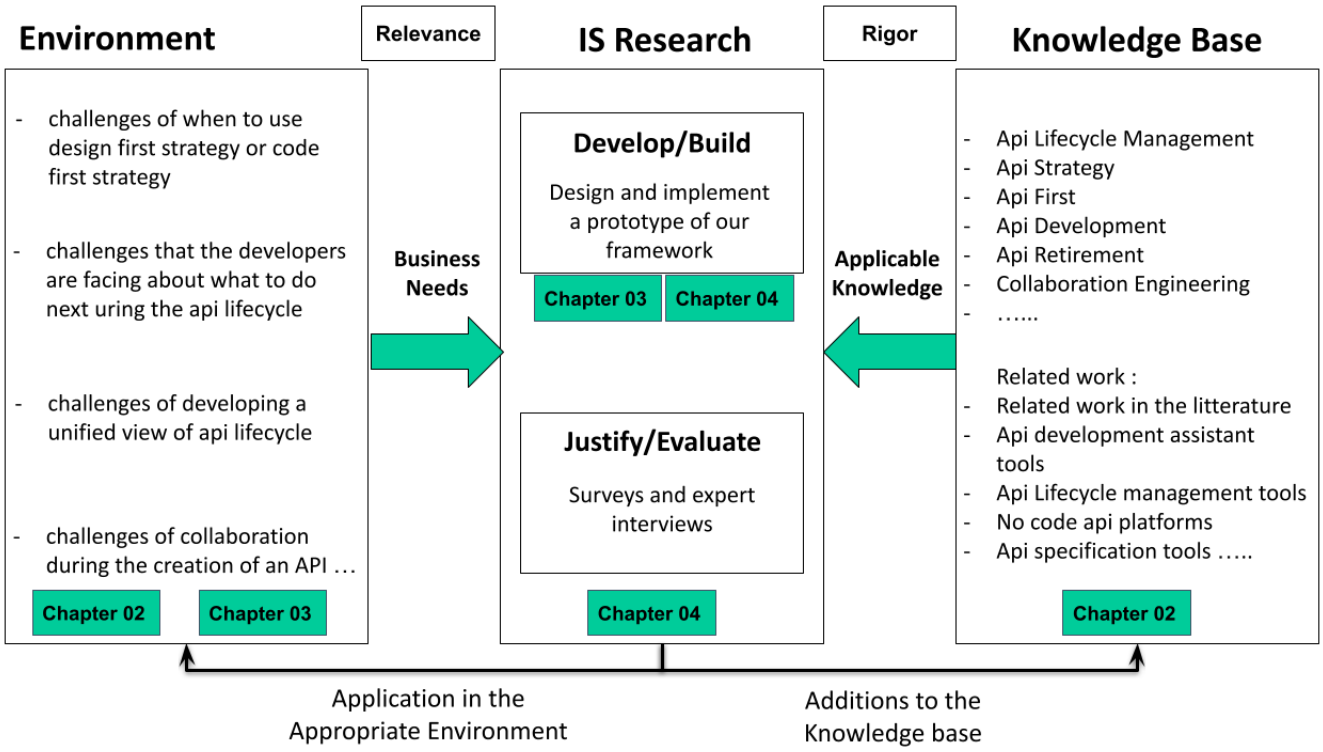

Figure 1.2 Information Systems Research Framework with Thesis Plan, adapted from [27]

Also, this framework that was mentioned [27] recommends following seven guidelines, which are as follows :

Guideline 01 : Design as an Artifact this thesis designs and develops a tool that will guide a small team or a developer during the API Lifecycle Management.

This API Lifecycle Management guide specifically solves a business requirement, that we faced during the development of StockAndBuy API, and the challenges that we collected from theoretical resources.

As described in 4, a prototype is implemented, designed, which matches and embodies part of the framework we worked on, as scientific investigation it is regarded as very interesting and based on our knowledge and experience, as well as the time we had available, but it needs more work and more improvements, and it is also a proof of concept.

Guideline 02: Problem Relevance in this thesis, the environment contains common problems from a literature analysis, which is to provide a unified view of the API Lifecycle, in addition to provide a guide for developers and small team for the API Lifecycle Management, with the aim of creating consumer driven APIs with a product mindset, in order to achieve this API consumers and providers must work together.

Guideline 03: Design Evaluation section 4.4 explains the evaluation method as well as the outcomes, in general, the assessment strategy includes a survey as well as interviews.

Our tool and framework outcome is validated only once in one iteration due to the lack of time of this theses.

Guideline 04: Research Contributions the first significant contribution of this thesis is the domain specific language of the API Lifecycle management, which was our path for creating a guide of a set of procedures that a developer or a small team must follow in order to create the API product that they want.

Guideline 05: Research Rigor in chapter 2 provide an overview of the foundations and associated research, a domain specific language for API Lifecycle management is developed, and a set of processes that a developer or a small team must follow, which was developed based on the acquired knowledge base and additional needs in the research domain.

Furthermore, the evaluation findings of our web prototype tool show potential areas for improvement, not just in the user interface usability of the prototype, but also in the framework design as well as the conceptual design As a consequence, the assessment feedback obtained can be used for future research, and iteration, in order to improve and refine the prototype and make it a real product.

Guideline 06: Design as a Search Process note that we previously stated in this chapter, that we need multiple iteration in order to create the tool and the framework that we really want, evaluation and refinement and more implementation and design is needed, because we only did one iteration.

Nonetheless, the evaluation data will aid in the redesign of both our framework as well as the prototype implementation, furthermore, it emphasizes constraints of the existing solution and suggests more future study, and a summary of all significant points.

Guideline 7: Communication of Research the overall approach and main conclusions of this master's thesis are delivered in a presentation at the University of Ibn Khaldoune Tiaret.

Another approach we used in this research was owing to the paucity and lack of documentation on the API lifecycle and how to create APIs, as well as current ways that allow this to be done more effectively, the majority of this material and references were posts or blogs in different websites and personal blogs, as well as videos and discussions on well-known sites as stack overflow and reddit.

So we began looking for anything that may support us or give us a cause to include resources like those mentioned earlier in our research, as well as a way to validate these raw materials, because the content of these assets will affect our research in a positive way, making our decisions more accurate and contributing to enriching the content of this study.

After a long time, the culmination of our research efforts was that we found this article [22], which shows how he used what he calls gray literature to do a systematic review, and after a deeper search, we began to find better results, and it was who said about gray literature [20] it is one of the resources released by new software engineering to share their expertise or information in the form of videos or blog entries, these items are not censored or reviewed in any way see figure 1.3 to get a basic picture of what we were discussing.

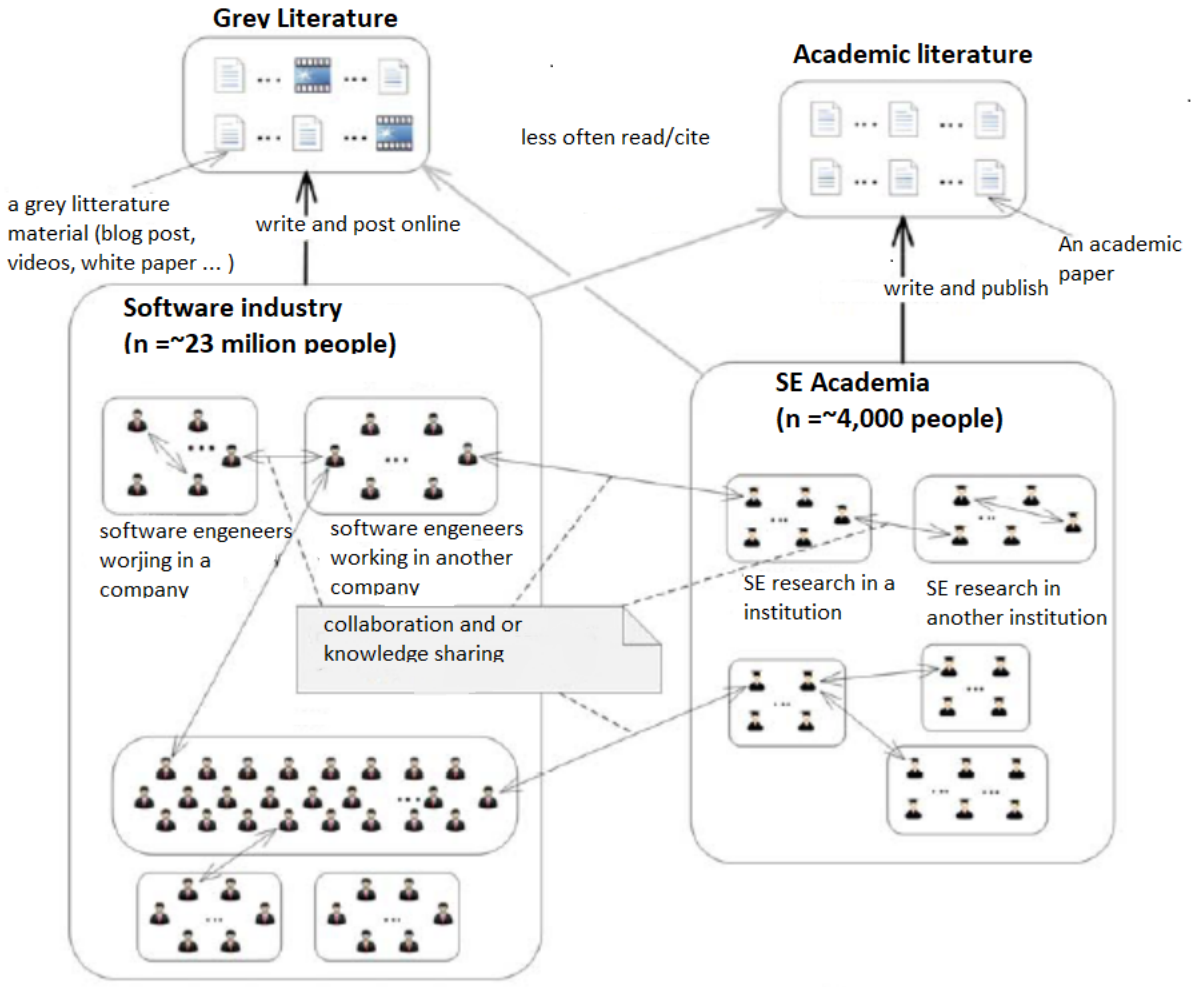

Figure 1.3 Visualizing the two communities (practitioners and researchers) and the knowledge literature produced by them from [20]

The article's [1] authors also divided gray literature materials into three tiers, the first of which has substantial tretriviability/credibility and is represented in books, book chapters, a wide range of periodicals, government reports, and more, then there's the second layer, which has moderate retriviability/credibility and is represented in news stories, annual reports, films, wiki pages, and presentations, as well as a third tier, which has low retriviability/credibility and is represented in blogs, emails, tweets, and catalogs check figure 1.4.

[1]

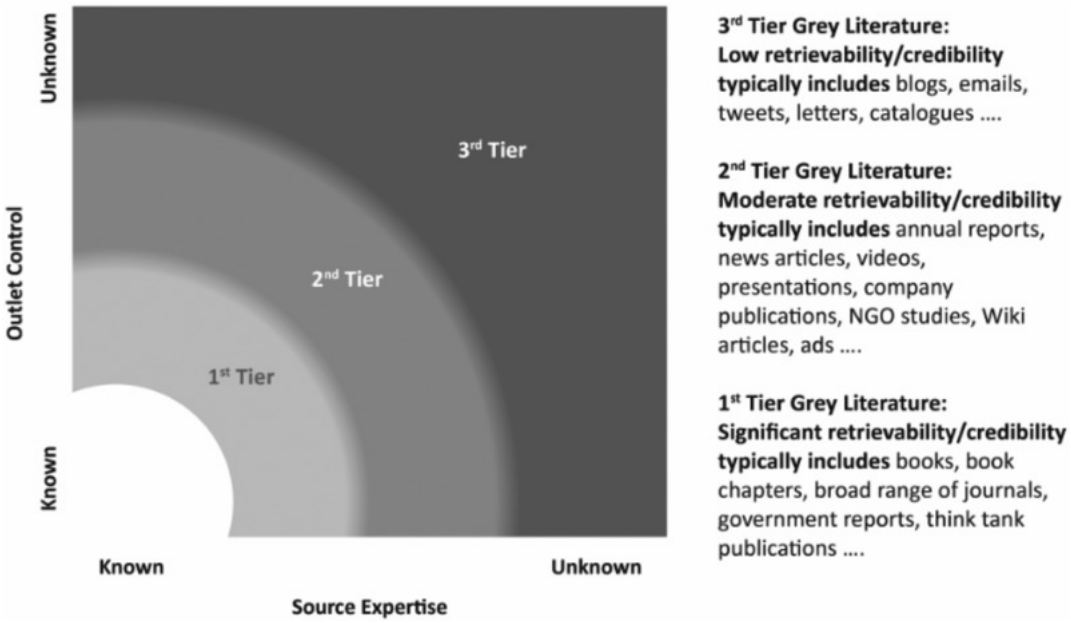

Figure 1.4 "Shades" of grey literatures (from [1]).

Let's also mention one of the most important findings of [1] research, which he did by analyzing 140 systematic reviews and categorizing them into three sections:

- A which included 44 systematic reviews, all of which used resources considered from gray literature.
- for the category B articles or the papers belonging to the mentioned last type has done an exclude of the gray literature, but it considered it as potential resources.
- And as for the one classified as C, no use of the gray literature was recorded.

It was determined that removing the gray literature from the papers and articles in class A would render them incomplete, and that this gray data was necessary in order to finish this research, which motivated us since we urgently needed to introduce the gray literature.

Finally in the last paragraph of this section, we'd like to mention the most scientific study that aided us in our work on the proper selection of resources from the gray literature.

[19] presented us with a set of principles and measurements to assist us in our decision-making, as well as examples of how to use these measures, these examples were really clear, and they greatly aided us.

1.5 Conlusion

In conclusion, everything we discovered and learned about design research in information systems and these frameworks has greatly aided us in attaining our intended objective, these frameworks have showed us what stages we must go through and what we must do throughout each step, which was an important factor and a support on which we relied while setting our objectives.

CHAPTER 2. BACKGROUND AND RELATED WORK

In this chapter, we evaluate works that are related to our own solution, and describe in depth how our solution enhances the state of the art.

We also provide an overview of key words, ideas, and technologies utilized in this thesis based on our knowledge base, which includes all we know about web APIs and their lifecycle management, as well as knowledge from the environment in the form of similar platforms, tools, and solutions.

We would like to share our experience with StockAndBuy company, because working with StockAndBuy had a serious influence on the content of these theses, it also shaped how we conceive about and envisage the area of web APIs, which we did not realize was so vast and had so many horizons.

We wrap up by describing how our study improves, in addition to this and after familiarizing ourselves with the field, we will assume the utility hypothesis as the starting whistle for what is described in chapters 1 and 2.

It is important to note that this chapter will offer information from both the environment and the knowledge base based on [27] and what [55] mentioned about design science.

2.1 web APIs Fundamentals

In this section, we will discuss the history of APIs, from the beginning of web APIs to how APIs contributed to the rise of cloud computing.

We will then move on to the key terms that developers, and those who typically create APIs use, because we will use them in the following chapters, and finally we will discuss technologies used in the process of creating APIs.

2.1.1 History

As we Mentioned in previously in this section 1.1 that Modern web APIs were theoretically born on 2000 with [18], for the industrial sector, Salesfoce may have been one of the first enterprises to recognize the value of this data-driven change according to [31],

Their web-based sales automation solution functioned similarly to the initial prototype of an API service, this application was built with XML since XML was hailed as the standard for transferring data over the internet at that time.

eBay too released the eBay Application Program Interface (API) [31] as well as the eBay Developers Program, which was initially limited to a small number of eBay partners and developers, Amazon.com web Services was established on July 16, 2002 [31], this enabled developers to add Amazon.com information and features into their own websites, as well as enabling third-party sites to search and display Amazon.com items in an XML format.

After this we saw Delicious, a new service for saving, sharing, and finding web bookmarks, debuted in 2003, using a web interface, but if you changed the extension from ".html" to ".xml," you'd get a machine-readable list of your favorites, Flickr, the popular photo-sharing website, debuted in February 2004, they released their API six months later, allowing users to simply embed their images on web pages and social media, flickr soon became the go-to picture platform for the growing social media movement.

Facebook debuted its long-awaited developer platform and API in August 2006, version 1.0 granted developers access to Facebook users' friends, photographs, events, and personal information, assisting Facebook in becoming one of the most successful social networks till today.

Google introduced a new Google Maps mapping system in 2006, six months later, responding to the numbers of rogue apps developed by reverse engineering and hacking of JavaScript applications, the Google Maps API became directly available.

In 2007, Twilio announced a new API as a product platform, this has launched a voice API that enables developers to make and receive telephone calls via any cloud service and builds

on the rising demand for speech-enabling applications, over the next decade, Twilio will become synonymous with vital resources on our phones such as speech, SMS, email, and others.

Ideas flowed until we hit on a concept that Amazon launched with Amazon Simple Storage (S3) and Amazon Elastic Compute (EC2), the first offers a basic storage service, and the second one provide servers that developers could leverage to deploy the necessary infrastructure for applications, this was a rallying point for transferring everything to the cloud.

After which it became clear that certain businesses needed to move to the cloud, and it was referred to as migrating to the cloud, and it became with regulations and standards after that.

It lasted several more years, but the population was overcome by social media by 2010, and APIs became a backbone, APIs became the way we linked and established company networks, exchanged photos and videos and told personal and professional life stories, depending largely on their communities, Facebook and Twitter dominated, third party developers and supporters to broaden their reach, increase their viewership and create a new generation of social impact driven by APIs according to[31]. .

2.1.2 Key Terms

After knowing the history of the literature, we can look back at some of the definitions we have identified for the API, and then web APIs, as well as the keywords that are used when creating the web APIs, we presented the tale before we specified the APIs, so that the reader can comprehend the future definitions and provide a broader image of what we want to provide next.

2.1.2.1 API

We can start by defining an API as “a way for two computer applications to talk to each other over a network using a common language that they both understand” [28], we also have the definition of [25] that APIs are the foundation of a digital transformation, they boost developer productivity by offering both programming freedom and a simple, user-friendly interface for accessing key functions, APIs are usually viewed as strategic assets by leading digital businesses,

created and maintained as developer-empowering products, APIs are viewed as middleware in the enterprise, as a means of integrating systems, or to reveal assets—it has the potential to damage almost all of its digital assets efforts at transformation

In MuleSoft [38] they define APIs as an Application Programming Interface, and it is a software middleman that allows two programs to communicate with one another, an API is used every time you use an app like Facebook, send an instant message, or check the weather on your phone, Mulesoft For individuals without a technical background, a real-life concept or example was used to help illustrate the API.

2.1.2.2 webAPI

web API is an application programming interface when the web is used as the network or as common language that the two computer applications both understand, web API is the main subject in our theses.

2.1.2.3 Collaboration Between API Stakeholders

In order to achieve a better results and a better web API which is going to satisfy the API Consumers collaboration is needed in every new API proposal, also in any new feature or change introduced in the API, you can read more about a collaboration framework for API Lifecycle management [9]

2.1.2.4 API Economy

One of the main point that teams focus on in the process of collaboration is generate a revenue, we can also say economy, they must first design a revenue generation plan before creating any API, the API Economy refers to this technique and everything that impacts it, this API Economy must be refined by collecting feedback from API creation stakeholders, tracking API usage, and developing an API market strategy, here are some of the key keywords influencing the API economy:

API Calls: A request to an API is referred to as an API call, a user's activity might be interpreted as a call, for example, if a user installs a new app, enters their information, and clicks the submit button, the program may transmit and retrieve data via an API request, HTTP queries are commonly used to access online APIs, although other protocols may also be utilized, the number of calls increases exponentially as more devices tap, from billions to trillions to quadrillions, this API calls can be managed by an **API Gateway**.

Continuous Integration: in order to manufacture the desired API product, and make customers benefits from the services of this digital product, a solid backend is needed, to produce this backend a development approach is used one of the most important approaches are continuous integration.

In his personal blog post [16] defined the continuous integration as a publication-based approach development known as CI, it enables team members to share thoughts with the rest of the team and observe how this changes effect them, it also allows the rest of the team to observe the direction of each member thinking during the development, when teams use CI, they may fail fast if there's an issue and this is the advantage of continuous integration, they'll notice it right away, usually within a few minutes.

We have another definition which is quite the same as the one mentioned above, in the late 1990s, eXtreme Programming promoted continuous integration, it is a method of software development in which modifications are incorporated as early as possible, and quite frequently, there is a vast range of options available, benefits of this method in allied fields, including as enabling agile testing and sharing the state of development inside the team, or boosting developer output, others argue that project predictability will improve as mentioned on [51].

Feature Branching Another approach, little bit different from CI, devefairly again he explained the difference between CI and Feature Branching in his blog post [16], he also gave a short description about feature branching by saying that most teams don't merge their branch until the feature that they are working on is complete.

Continuous Deployment An ancestor of CI is CD or continuous deployment, in [49] researchers defined CD as a collection of procedures intended to make the transition from version control to production or release to manufacturing as smooth as possible, automation of the test and deployment processes, and the use of continuous integration to quickly evaluate the accuracy of every change by executing the automated build and test process are all important components in continuous deployment.

DevOps and of course after talking about CI and CD we should move to devops, because this two are used a lot during the process of applying this methodology, in [8] Ineta Bucena and Marite Kirikova said about devops as a way to deliver software faster and a way for a wider collaboration between team members, allow bringing agility into the maintenance part of software development process.

API Gateway: Managing an API necessitates the use of software to consume requests and process them according to a predefined architecture.

The gateway is the system's entrance point and conveys what the user wants to the database, a great API gateway design identifies the optimal path to the data and determines how to translate it for the end-user.

The gateway accepts the request, finds the optimal route for the function, integrates any required services, and provides a consistent experience, throttling, rate limitation, and authentication may also be handled by an API gateway.

Since a user might access their account via a cellphone, computer, or smart TV, the gateway determines what devices are connecting to the service and if they're authorized before pulling up the best experience possible for the user based on the platform.

Managed web APIs: Because one of the primary functions of an API gateway is to manage APIs, according to [30] managed web APIs in more detail, managed web APIs need the client to submit an API Key, which must be supplied as part of every request, API Keys are needed, providing a large number of prominent APIs, including those offered by organizations Google, Twitter, eBay, Netflix, and others Facebook.

According to research conducted in 2012, there were at least 10 API providers who handle more than one billion requests API requests are made each day and must be controlled, API Keys are generally used to verify the user's identity, and/or identify the client system, allowing many managerial duties,

Idempotency: essentially means returning the same or similar results across devices and requests, APIs should present consistency for the user, it is related to commands tied to HTTP and servers, if someone punches in the same thing more than once, they should get identical results every time, that includes visiting a website from Chrome and later through Safari.

Idempotency is associated with http verbs such as POST and GET, the function of these verbs is to assist us in generating or retrieving data; these verbs are referred to as **CRUD** operations, CRUD is an abbreviation for Create, Read, Update, and Delete in computers, CRUD refers to the various functions that an application must be able to do.

An app, for example, must generate new entries, read current entries, change old options, and remove things, most programs handle CRUD capabilities at the most basic level, if not, the program most likely does not provide complete read/write access, in order to perform this crud operation we need links so i think its the perfect time to talk about hypertext as the engine of application state or just HATEOAS.

HATEOAS: as [30] mentioned in his article hypertext as the engine of application state is an essential component of the human web, people follow links; each link refers to a visible resource, and links that are not legitimate in the particular scenario are not included, and therefore the client does not interact with them.

The client just sees the valid operations, in contrast, programming interfaces have typically offered a predefined set of operations, and the client must be built with the knowledge of which actions make sense in each particular application state.

2.1.2.5 API As Product

APIs, or application programming interfaces, are the means through which software communicates with other software.

However, APIs are much more, they are interfaces that let developers to construct new goods and services by frequently using data, functions, and applications. In [24] when they want to show why API is product they started with an example of when a ridesharing app receives an order for a car, it uses services such as Google Maps APIs for navigation, as a result, such ridesharing firms express their operations using APIs.

Also according to [24] APIs aren't simply some back-office technical detail; they're both goods for the developers who create today's consumer experiences and the means through which value is increasingly shared in modern economies.

2.1.3 Technologies

Including messaging systems, architectures, protocols, and API specification standards this all are technologies used in the development and creation of web APIs, figure 2.1 illustrate the history of technologies used in developing web APIs.

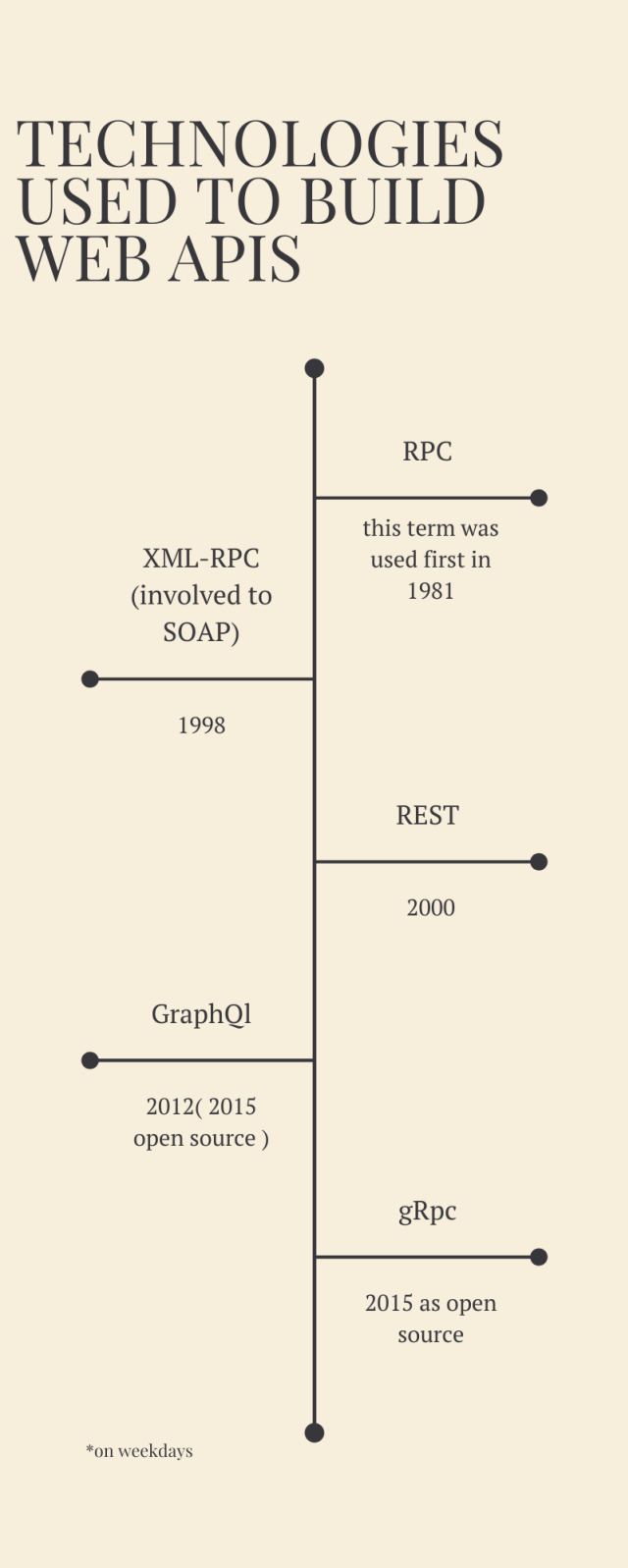

Figure 2.1 History of web APIs technologies

Messaging systems, in general, are concerned with message transmission, which obscures the intended functionality and behaviors of clients and servers this as commented by [30]

2.1.3.1 RPC

Implementations of RPC (Remote Procedure Call) encapsulate all communication code, making communicating applications easy to comprehend, this method of concealing complexity is often used RPC has some issues related to that RPC is a tightly coupled which is going to lead to a lot of problems when a one is updated, another issue is that RPC done show network communication so developer will ignore some potential network failure and a lot more issues which is mentioned on [30] in the RPC System section,[30] also mentioned that the issues with RPC was later solved by CORBA.

2.1.3.2 XML-RPC

It is based on RPC a return data in a form of XML, XML-RPC later standarized as SOAP, the Salesforce mentioned in the history subsection 2.1.1 was based on XML-RPC.

2.1.3.3 Rest

Based on what [40] said, REST is the least-used API paradigm — this is just a small proportion of APIs produced despite the fact that more broadly the word REST is abused, clients do not generate URLs, this is a characteristic feature of this sort of API; they just use the URLs provided by the server as they are.

It does not collect the URLs it uses and does not understand the website form of the URLs used; it follows just the URLs that it discovers on the pages received from the server, the browser also operates like this, this functions, robably the most popular APIs now a days is using what we can call Restfull which based on REST and RPC and uses http as protocol to communicate the author in [40] is calling it OpenAPI.

2.1.3.4 OpenAPI

From the official OpenAPI documentation [2] they call it OpenAPI secification (OAS), which define a standard language to deisgn interfaces for Restfull APIs and according to [2] helps people and machines to learn and grasp the capabilities of the service through network traffic inspections without access to source code, documents, when described correctly, the customer may comprehend and interact with the distant service with a minimum logic.

2.1.3.5 GraphQl

GraphQL was internally developed in 2012 by Facebook before its publication in 2015, the GraphQL project transferred on 7 November 2018 from Facebook to newly created GraphQL Foundation, housed by the NON-Profit Linux Foundation, GraphQL is a web API open-source data query language and runtime for queries, there is also another technology used in a service to service communication, called gRPC.

2.1.3.6 gRPC

Google first built gRPC which uses Stubby to link a large number of micro-services in and across its data centers using a single RPC architecture, google agreed in March 2015 to create and open source the next Stubby version.

2.1.3.7 Git

To build the desired API using the mentioned technologies we need a way to control the updated and the versions of this API, and one of the best tools to do this is the version control system git, a software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development, its goals include speed, data integrity, and support for distributed, non-linear workflows,

2.1.4 Code Collaboration Or Review Platforms

This is usually paired with git or other version control systems, provides a way for a developer who isn't the author of the code to study it, code review and code collaboration software allows several developers to observe modifications to other developers' code, lowering the chance of bugs, security flaws, and requirements that aren't met, software development teams employ code review software during the development phase of a project because it is less expensive, faster, and easier.

2.2 API Lifecycles

From the following reports Gartner Magic Quadrant for Full Life Cycle API Management [44] and The Forrester Wave™: API Management Solutions, Q3 2020 [47] by Gartner and Forrester. we collected 20 view of the API Lifecyle we had a chance to review 11 API lifecycle view we will explain it here :

2.2.1 Axway

[4] see that The management of the API life cycle is the necessary stages for the life of an API from creation to retirement and the challenges while building an API are Creating APIs, Controlling and here we can talk about security which is the most important part in API controlling, Consuming.

2.2.1.1 Creation

The **Creation** part contains processes like : Model (visual / programming): specify the data required for your API endpoints.

- **orchestrate** unify data format, which we get from many sources.

- **trasform**: convert legacy format such as xml to contemporary forms such as json (Sometimes we have xml API to be exposed to json, then to the customer so we need a mapper).
- **document**: auto generates exemplary doc and code for models and API activities automatically.

2.2.1.2 Control

The **Control** activity composed of :

- **deploy** instantly with a, zero setup and effort, no time between code delivery and deployment so we need to use devops tools with ci/cd, for small project you need to come with a solution when you can deploy to with a single click.
- now we can add **manage** to control the access to our API via rate limiting, when you have a lot of APIs you can not manage them you will need an API catalog to manage : publication state including unpublished, published, deprecated, retired plus versioning and rate limit to protect the system from high traffic usage.
- **secure** or security and fire walling APIs, its a paramount API management solution to offer a top notch API security, security certification such as common criteria is also strongly recommended.
- **scale** infrastructure up or down to run your server side apps, automatically adjusts with the traffic with no manual interaction.

2.2.1.3 Consumption

Finally we can discuss the components of **Consumption** which is :

- **publish** market to internal groups, partners, or the public via a central catalog to an API developers portal to be easily consumed by them, internal, public or private catalog are managed differently and require to have a partner on boarding process on the solution.

- **marketing** should not be neglected for example open or public API need an attractive branding and also promoted through the different marketing channels.
- **invoke** execute API operations, can generate SDKs from API developer portal in their favorite language.
- **monetize** track utilization and apply rate plan policy to generate API revenue, here, axway adds another service, which is analytics, and says that it is in all the previous parts, and they justify this by saying that you cant control or improve what you cant measure, this figure show how Axway see the API lifecycle 2.2.

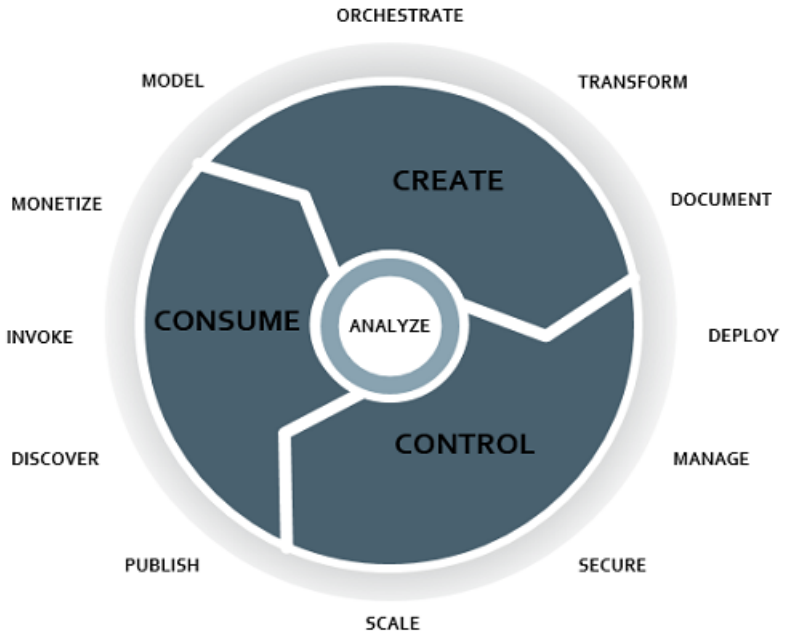

Figure 2.2 API lifecycle according to [4]

2.2.2 Redhat

in [35] define a two actors **Producer** and **Consumer**.

2.2.2.1 Producer

- the producer can set **goals** including strategy define the goals, market to address, resources you have, time frame to achieve, help put the effort in the right place.
- **design** using design first, gather feedback, reduce risks, reduce time to market, design an API and share it with future consumers, API contract that describe the messages that can be exchanged with your API, the producer can specify including mocking you need to specify a meaningful payload examples, refine the specification with the business expectation, live mock can be exposed to consumer.
- in addition to **business** expectation entail everything that can not be expressed in a formal schema.
- **testing** if your using the test driven approach you need to write tests based on the examples before implementing the API.
- develop and deploy or implement, in this phase the API is developed using either an integration framework, or programming language, for an integration framework it can promote reuse, fast iterations, and value additions through orchestration, modernize the existing parts of your information system.

  The usual agile development practices apply here, modernize the existing parts of your information system, we can add deploy and making sure you have CI/CD pipeline to automates the delivery of your APIs to the production environment., use the tests that we wrote earlier to ensure the API to be deployed, and not violated, those integration tests are automated and run automatically as part of the pipeline to ensure backward compatibility.
- **secure** a key step, static analysis, vulnerability testing is a part of the CD pipeline.
- **manage** after the development and security versioning, semantic versioning minor version replace the previous one and then consumer will switch, breaking change will lead us to a

major version to be released deployed side by side with the previous one, consumer migrate by adapting their code to the new version.

2.2.2.2 Consumer

We now move to the **Consumer** he is not a real actor the activities in this phase need an active collaboration with the consumer, and that's why we are including this phases here here red hat is mentioning steps like make it available for the consumer to easy discover this API, develop, the right way about how to consume this API in addition to a strategy to refine the API including monitoring and refinement, check figure 2.3 to see a global view of the API lifecycle according to RedHat.

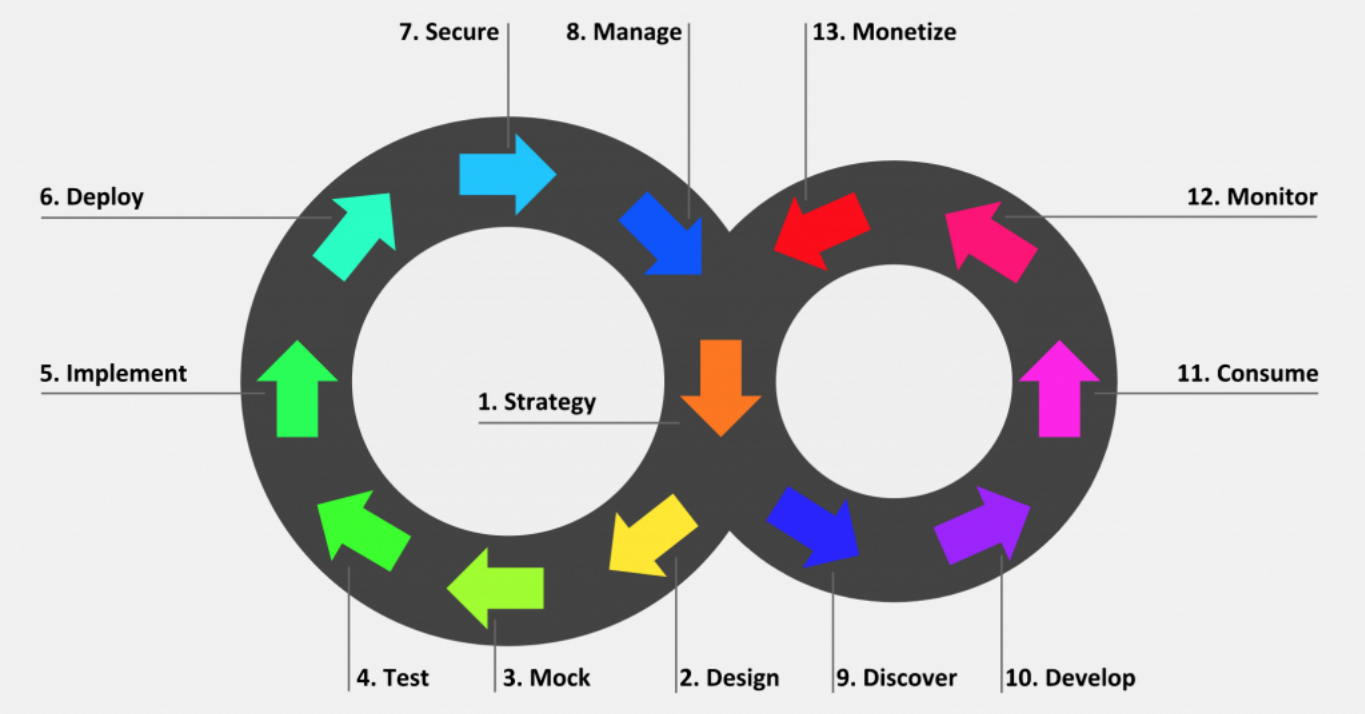

Figure 2.3 API Lifecycle according to [35]

Google cloud platform or APIgee: Include rich resources for educating customers prospects on API business potential risk analysis on API calls According to [47] and [44].

2.2.3 Software AG

We saw in [47] and [44] that customers are able to adopt their solutions by various paths, support gateways from other vendors, API hackathons and beta programs, expanded ecosystems, API product management, multi cloud and edge computing, and extend container and service netting environments to customers for more combined management, monitoring and governance.

For the most part customers can use API strategies, the solution is versatile enough to enable a broad range of API strategies for API design, policy management, federated API publication and analytic, especially to clients who are ready for strong governance and discipline to assure the strategic success of their API programs.

2.2.4 IBM

Gateway architecture, API design, and multi cloud functionality, API non-rest, integration, API monitoring, federated API publication support, DevOps, API product and flexible API documentation. API federation support use the solution in every category of API: internal, B2B, open web, and product APIs, showing how a broad range of API strategies is delivered through this solution, check [47] and [44] for more details, and check figure 2.4 if you want to know how IBM see the API lifecycle.

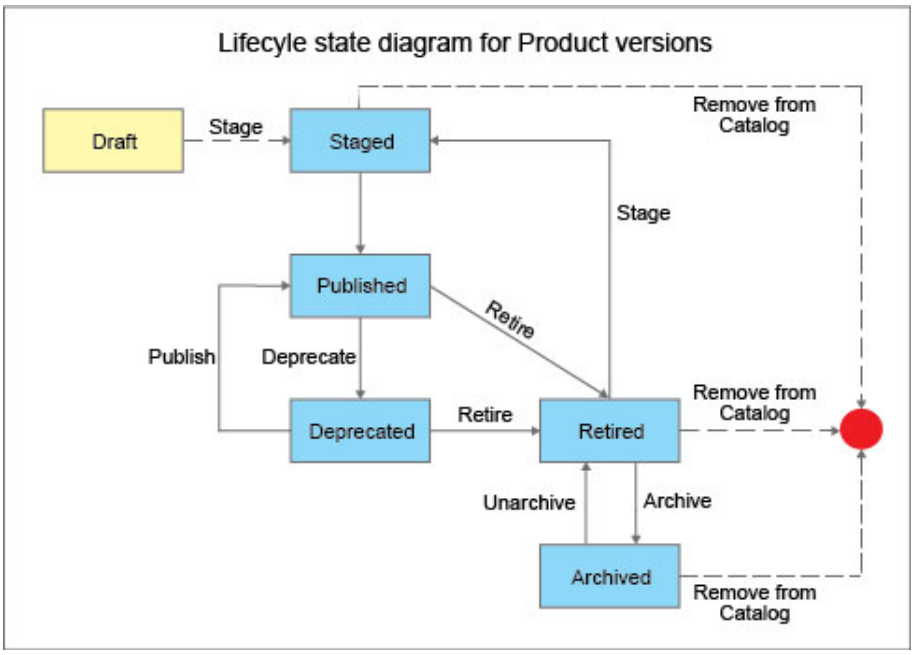

Figure 2.4 API Lifecycle according to IBM

2.2.5 WSO2

Open source solution, free forever API management path, deep microservice integration platform, graphql support, excellent packaging of API products, API integration, versatile and configurable user portals, CI and CD (CI / CD) open source solution in particular, the solution is ideal for purchasers with integration plans, API's, and microservices combine with an open source need for advantages, as [47] and [44] commented.

2.2.6 Sensedia

Control system who are also seeking strategic guidance in API programs and business strategies, along with a combined product line for APIs, microservices and event architectures, API business strategies, API programs and governance, inclusion, life cycle management, and lower non rest API and API product performance you can find all of this on [47] and [44].

2.2.7 MuleSoft

The excellent for buyers who need the integrated integration and API management skills of the API user commitment, REST API design and proxy, delivery management, analytic and partner federations, may augment the assistance of its API strategy team, more about this on [47] and [44].

2.2.8 Kong

Open source service mesh is a highly customizable administration of APIs and for people who are looking for basic management capabilities in addition to Kong Gateway installations, according to [47] and [44] for.

2.2.9 Microsoft Azure

Provide consumers the fundamentals and a way to a customized, self-managed environment, the main limitations are restricted user interaction capabilities for APIs, limited search and browsing for APIs, no integration of micro services, and limited API policies. Azure platform purchasers that either have a basic API administration needs or aim to substantially adapt their API user site.

2.3 Related Work

2.3.1 Academic Part

In theoretical part, based on our knowledge and experience, as well as the time we had available for this theses, we could not find a lot of related work, and one of most related work to us was what Duc Huy did in [9] his aim was to create a collaborative approach for API life cycle management with a support tool this comprises the collection of criteria for a small artifact, a conceptual model for the API lifecycle, from literature and expert interviews, a case study with expert interviews evaluates the Design artifact, the outcome demonstrate that a prototype is a workable solution.

2.3.2 Industrial Part

Here we want to inform you that all the API leader companys and organizations that were mentioned in 2.2 are included in this framework and that they all aim either to facilitate or replace all API management lifecycle.

We have identified other websites and tools in our study, the platforms we did not include in 2.2 are the following :

Microcks is the open source Kubernetes-native tool for API mocking and testing that helps you during the ideation and prototyping phases.

APIcurio helps you collaboratively to develop better API contracts, API designs may be worked jointly by architects, product owners, designers and developers.

FireAPIs No-code API builder platform to develop an API in minutes without worrying about installation, scalability, availability.

2.4 Conclusion

In this final section and after seeing all of this related work and similar ideas and platforms, some of these platforms were too sophisticated or too complicated for one user or small team, while others needed a payment to benefit from.

This was one of the main reasons that made us work on our framework that we want to call *COVERING-API* and its support tool.

After reviewing the literature to get more information about the state of are in the domain of API lifecycle management in the academic era, we want to propose a utility hypotheses, because we are following the framework of [54] that tell us how to do theory building in the design science researches.

Our utility hypotheses was if our framework *COVERING-API* is used correctly, developer know what to do next so he can deliver an API faster and a team can collaborate better during the development.

CHAPTER 3. DESIGN OF OUR FRAMEWORK

This is the opening paragraph to the third chapter of this thesis, after defining our research approach, taking a tour through the literature to build theoretical foundation, this was followed by a review of how the industrial leading company and organisations see the API lifecycle, we then formulated our utility hypothesis forming a bridge [55] between the the solution concepts and our understanding of the problem. In this chapter we will explains in detailed terms the concepts of our API life cycle management framework and the steps that we followed to design and build our framework which is intended to guide you through the life cycle of the web API.

3.1 Challenges

Based on our experience we divided this challenges into two parts, the first part gives us a brief overview about the challenges that we faced while working on the StockAndBuy API which is under the title of industrial challenges, the second one is named market research and its based on the related work section 2.3.

3.1.1 Industrial Challenges

During our internship and while we were working on StockAndBuy API We encountered many challenges, we identified 5 of while we were working on the API, for the other 3 challenges we added them after a long thought about the reasons for our failure to complete the project as we planned it, and an analysis of what happened during that period of internship. Note that this 8 challenges is only related to API development and API lifecycle.

- challenge 01: Was the spark that ignited the idea of *COVERING-API* framework, we were unfamiliar with the overall and different steps and stages of the API lifecycle management

process, so we had to search and read a lot about the best way to do a certain step in this life cycle, then find and determine what to do next, which was a major issue.

Additionally, StockAndBuy customers wanted the API as soon as possible, but the searching has taken a lot of time and more than we expected, this had a big part in our failure to deliver the product as intended; throughout the various sections of the chapter dedicated to project management, we will explain why we opted to approach this project in this manner 4.5.

- challenge 02: Difficulties to choose between code first and design first methodology, we needed some metrics to base on and to be able to make the best possible choice, which leads us to go to the coding stage directly.

- challenge 03: We used a version control system called git with the code collaboration azure DevOps provided by Microsoft, we created commits every time we completed a key task, which is similar to taking a snapshot of your repository.

  The main reason for writing commits is to look at the changes history when we need or want to know what we were working on at a certain point in time.

  Another reason is to generate an understandable and easy to read changes log from these commits, which will be difficult because we weren't following a commit standard.

- challenge 04: Another issue is one of monitoring and arranging tasks; the API product manager must keep track of what we're doing and assign a time to each task to guarantee that the project is well-organized.

- challenge 05: We had a hard time figuring out what the customer needed since we couldn't figure out what question to ask him.

- challenge 06: Due to our lack of experience with API lifecycle management, we were unable to determine whether the API should be public, private, or only for partners.

- challenge 07: It is shown that the deployment of changes to your web application several times per day helps teams enormously to create better products faster.

  It is an important element to frequently deploy to check if the changes function and confirm that the major problem is solved, it allows developers to generate a preview of how the production will operate without releasing anything not ready, but this is not something that we have given much attention.

- challenge 08: The need to invest in APIs that allow our customers to create valuable workflow, APIs that return data but are difficult to incorporate into an end-to-end workflow aren't very useful.

3.1.2 Related Work And Literature Challenges

In addition to the issues arising from our StockAndBuy experience, our literature analysis and comparable initiatives and a platform identify certain issues, the fields discussed include:

- challenge 09: What is the greatest development strategy to use after adopting a design-first approach TDD (test driven development) or writing code first

- challenge 10: Many developers confuse feature branching methodology with Continuous Integration as Dave Farley said in [16] and [17].

- challenge 11: We discovered a need for a unified web API life cycle after analyzing and reviewing each of the leaders view to the API lifecyle in the following reports Gartner Magic Quadrant for Full Life Cycle API Management [**?**] and The Forrester Wave™: API Management Solutions, Q3 2020 [**?**] by Gartner and Forrester.

3.2 Requirements

In this section a several and different requirements of the *COVERING-API* are collected and presented, we chose to divide it into two major parts functional and non functional

requirements. But first we want to show the challenges that we faced, which made us think about creating such a framework.

3.2.1 Functional Requirements

Knowing that we had left 05, 08, 10, 11 challenges to be utilized in more research and the next iteration to strengthen our framework.

The remaining foregoing challenges resulted in the following functional needs:

- R01: The creation of a process or a path allows a small team or a single developer to manage the API lifetime and to know what to do in the next phase.
- R02: Developers need metrics to decide whether to use a design-first or a code-first approach.
- R03: Developers should rely on standards for writing commits to generate a better change logs.
- R04: A metrics to decide if the APIs will be public, private or for partners only.
- R05: Create a tool that can track what the team is doing, to generate a task board change log.
- R06: Deploy the changes continuously to a staging environment using a CI/CD pipeline.

3.2.2 Nonfunctional Requirements

- R07 (Performance): Our system must respond fast to a certain user actions, usual delay time should not exceed 10 sec in most conditions, as indicated in [39].
- R08 (Usability): We must provide a framework expressed in the form of diagrams that are easy to read and understand for anyone with programming experience, with the goal of developing web API.

Based on this framework, a tool must be developed, with the first goal being an easy to use user interface (UI), after meeting the functional requirements.

- R09 (Portability and compatibility) : A web cross platform, cross browsing and mobile responsive solution must be provided.

3.3 Conceptual Design

Since this theses have led to the development of the framework called *COVERING-API* (Conceptual Framework For API Lifecycle Management), we first give an overview of the capabilities that it provides. *COVERING-API* offers services for API Life cycle management dedicated for small teams and developers that are comprised of members made in a task collaboration space (see figure 3.1).

Our framework offers two main services: (i) API lifecycle management assitant, (ii) ensure synchronous collaboration, and, our proposition is based on Business Process Repository.

Hence, we take benefit from conceptual modeling which is a description language dedicated to express APIs Management process formal abstractions, also, we use model repository to ease the management of conflicts related to the collaboration and versioning.

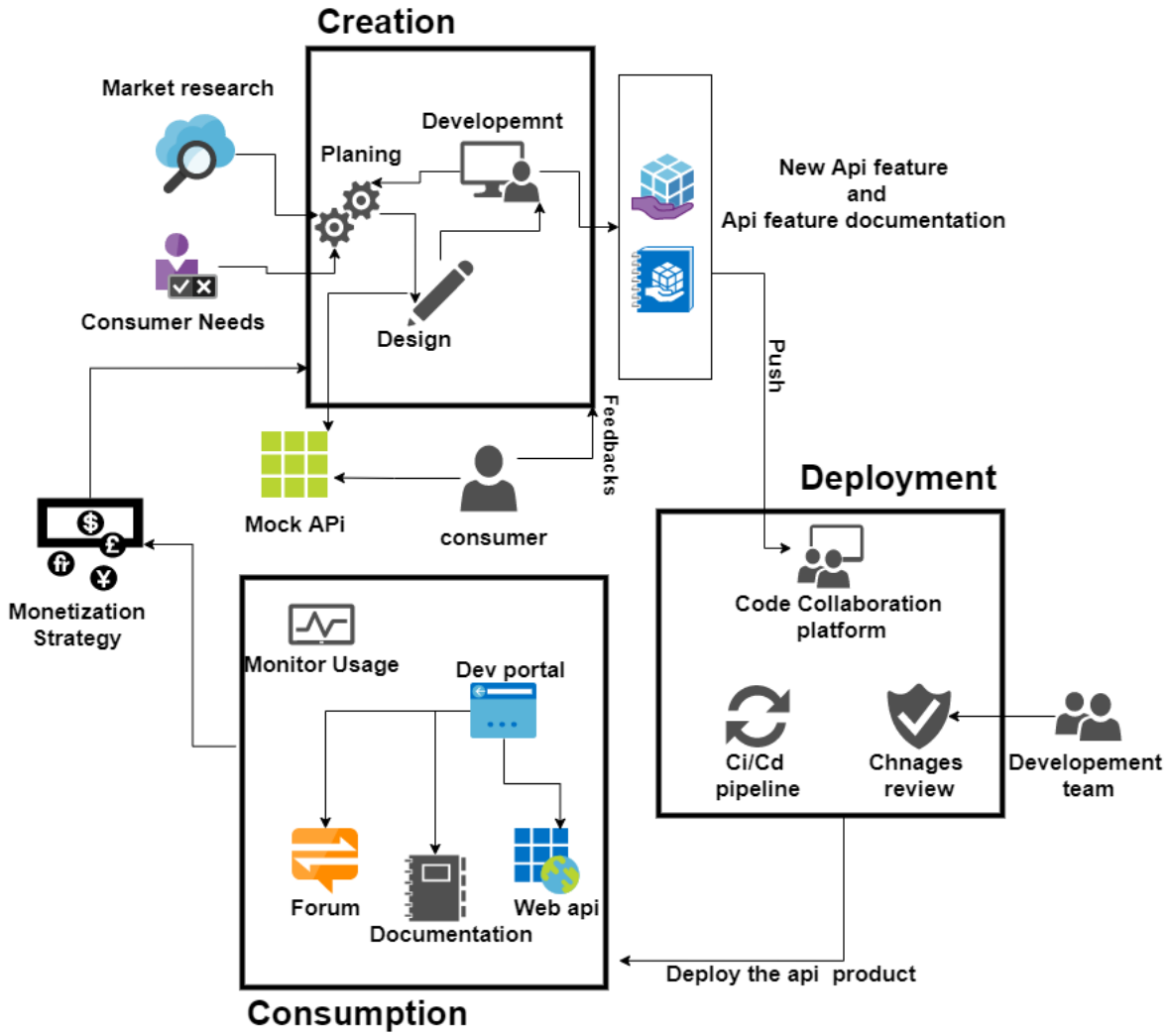

Figure 3.1 COVERING-API Overview.

3.3.1 APIs Management process through a service based pipeline

We offer three primary API stakeholder services in this part: Creation service, Deployment service and Consumption Service.

3.3.1.1 Creation Service:

This service reflects the system's capability in providing a guideline for a unified view of collaboration, communication and analysis during the API lifecycle journey thanks to our DSL (domain-specific language), which was developed in a way that aims to find a generic meta model of this lifecycle, with the help of a WWW question answering system.

Our DSL have a large influence in the transformation of consumer or customer needs into links that provides data and services.

3.3.1.2 Deployment Service:

We wanted to make the outputs of the creation service available for all stakeholders, we also want to mention that collaborative engineering research show that while the team goes through a multiple iterations of reviewing and refinement of the product an easy and scalable strategy to deploy and share artifacts and knowledge is crucial in this phase. Moreover this is a significant determinant of success in every project see [46] and [3], this is one of the main aspects of devops see [41].

3.3.1.3 Consumption Service:

The aim of this service is ensure that consumer needs and requirements are deeply collected furthermore its the best phase to improve the api product strategy, and to provide the best experience for the customers when the API available, it is closely linked to the deployment service. Finally i want to mention that the main focus on this three services is to ensure interoperability, reusability, discoverability, governance, modularity, and distribution this 6 are one of the fundamental priciples of service oriented architecture (SOA) see [23].

3.3.2 Conceptual Organization of *COVERING-API*:

The meta-modeling attempts to bring out concepts, attributes and relationships that determine a given domain through a meta model [10].

One of its main goal, which has motivated this choice, is to meet the constraint of context-aware systems, we have proposed a design languages for *COVERING-API* as an intermediate framework between API developers (testers developers, designers).

This framework is based on the *COVERING-API* language which is conceived as a DSL for API lifecycle management system, see figure 3.2, Dev-API language is composed of a set of packages see figure 3.2, where "'Planing"', "'Design"', "'Develop"' "'Deploy"' and "'Publish"' packages are used to define the different phases of the API lifecycle management.

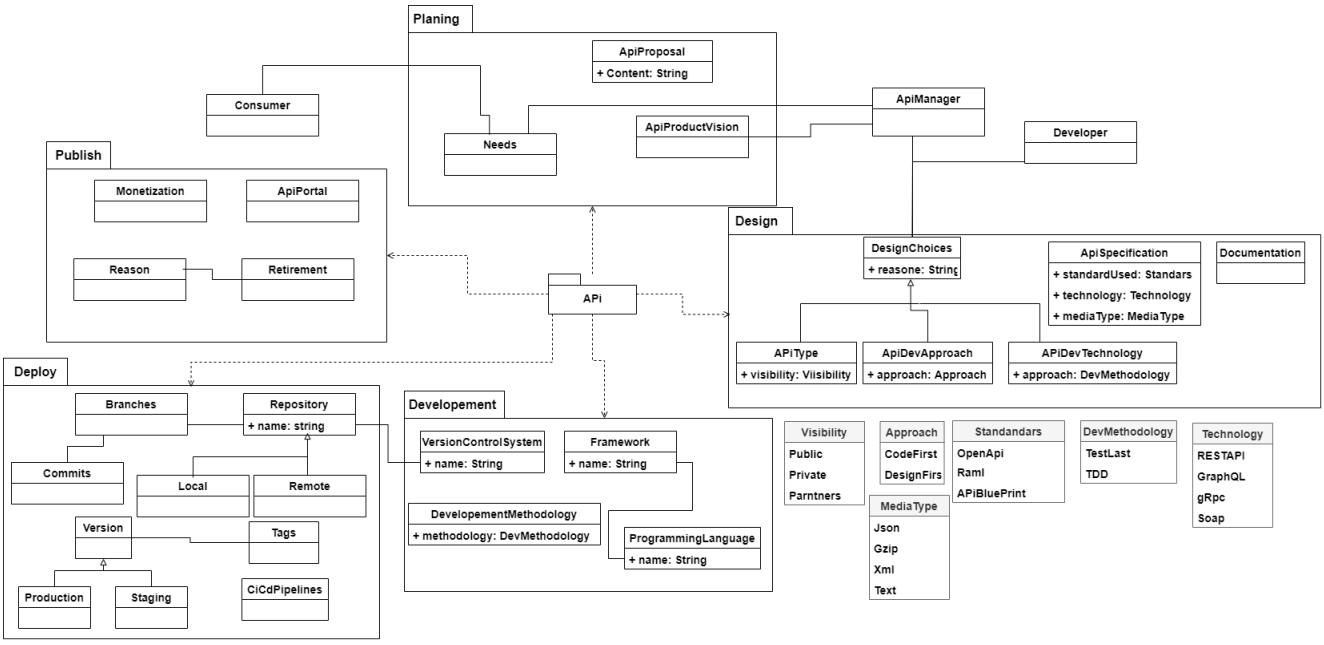

Figure 3.2 General structure of *COVERING-API* language.

"'Planing"' package represent the functional side of the system, while the "'Design"' and "'Developement"' packages provides the non-functional side of the system, when the "'Deploy"' and "'Publish"' packages, they provide a way to share the work that each member has an impact on developing the API does. The first package is concerned with providing a seamless process so that the team can deploy the API, do the versioning process, and work on new features in a clear

and organized manner, refine the API Monetization strategy, an API portal that we can think of as interactive bridge between API providers and API consumers.

Retirement have reasons we consider it as one of the most influencing sides of the publish package. To considered a variability in customers' needs as a set of features as their mandatory, optional, alternative, requires, and excludes relationships that represent a characteristic of a system.

3.3.3 Process of our Framework

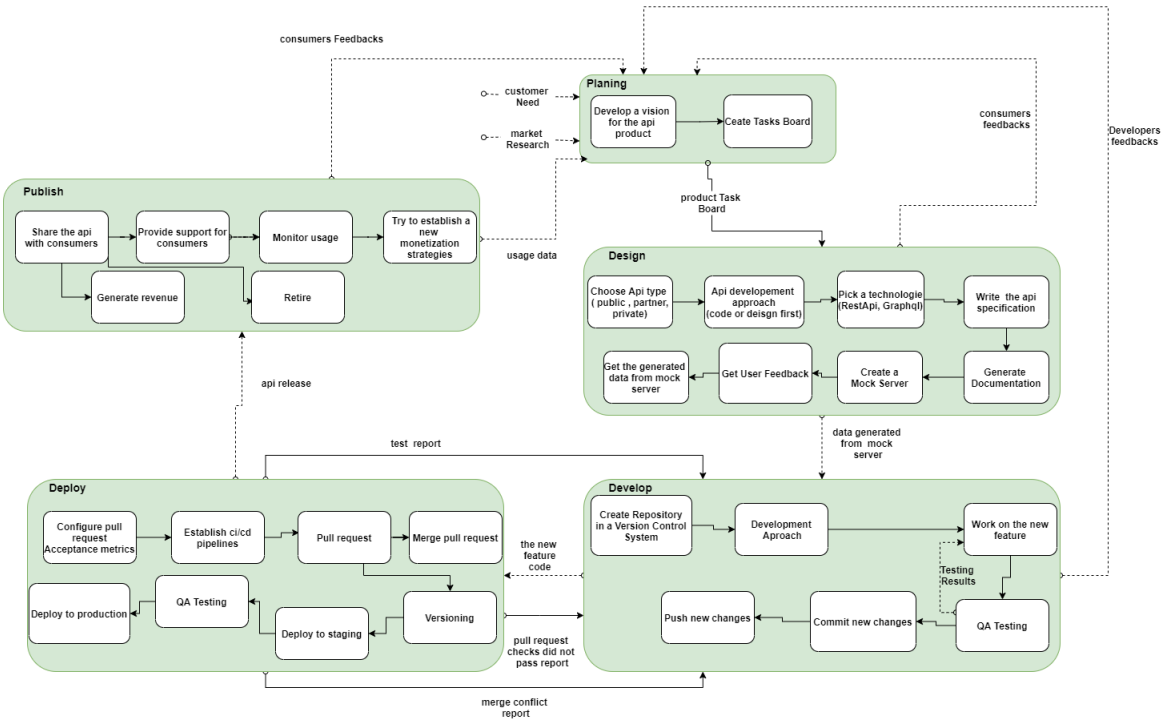

Figure 3.3 Process of *COVERING-API* Framework.

3.3 shows the process of our framework, our contribution is based on its core components and the different links between: Planing, Design, Develop Deploy and Publish, packages see figure 3.2.

Users need to manage API lifecycle in a given project to create a complete and a use full API product that the consumer can benefit from its usage, this proposed approach is a multi-step process primarily consists of 9 phases:

1. During the First phase, the user of this framework need to do a market research and collect the costumer needs, based on the data collected and his (API manager or developer) goals he will develop an API product vision.

2. During the second phase, user starts by creating a task board, this can be his own task board he is alone or for an entire team.

3. In beginning of the design phase, this user have to choose one of the three API types: public, partners or private API this is will depend on the information collected in the planning phase.

4. In this stage our user need to choose between code first approach or a design first approach, this is based on the previous stage.

   If the API is public then the best choice is design first, and this is based on our research and our follow-up on what the API development leaders are doing, if this is not enough to choose design first then the user have to answer some questions (e.g. "'API is the main product for a business ?"'), but if the API is a partner or private API then the choice will depend on the last three question and the team choices.

   As for code first approach becomes an option when the team want to deliver the API faster or when the answer of last three questions is no or if the API is a private API and the user have to go directly to the development phase.

5. The 5 phase is about choosing either REST API or GraphQL this is also depend on last two stages if the API is public and the user has picked design first then the best choice is REST API, as for GraphQL, it is always an option and it depends on the team choices.

6. In a 6th step, and if the user picked the design first, its time to write the API specification from this specification you need to generate the documentation, create a mock server and share it with the team members and API consumers in order to get feedback from them.

7. In a 7th step, we are now moving to the development phase, the development team need to pick a version control system and code collaboration platform and start by choosing between a writing code first or a test driven development methodology, after this the development team need to start working on a new API feature or fix an existing issue in the API, when they are done with this they need to make sure to write the required tests, and that this tests is passing, this tests is important because they are going to play an important role in the next step and exactly in the CI/CD pipelines.

8. This step is announcing the end of the development stage and the start of the deployment phase, where the user and his team need to establish a CI/CD pipeline in order to check the other team members pull requests against the main which must be restricted, so no one can merge his pull request before the review, the number of the required reviews to merge is determined by the team members and check passes, checks is a CI/CD pipelines to check the code coverage, if the pull request commits is following a commit standard like conventional commits for example, and to check if the unit and integration tests has passed successfully, or to deploy to a staging environment, take a place, in the end the API product owner, or the API manager will create new tag each tag is a new release the name of this must follow a version naming standard like a semantic versioning or date based versioning see figure 3.4.

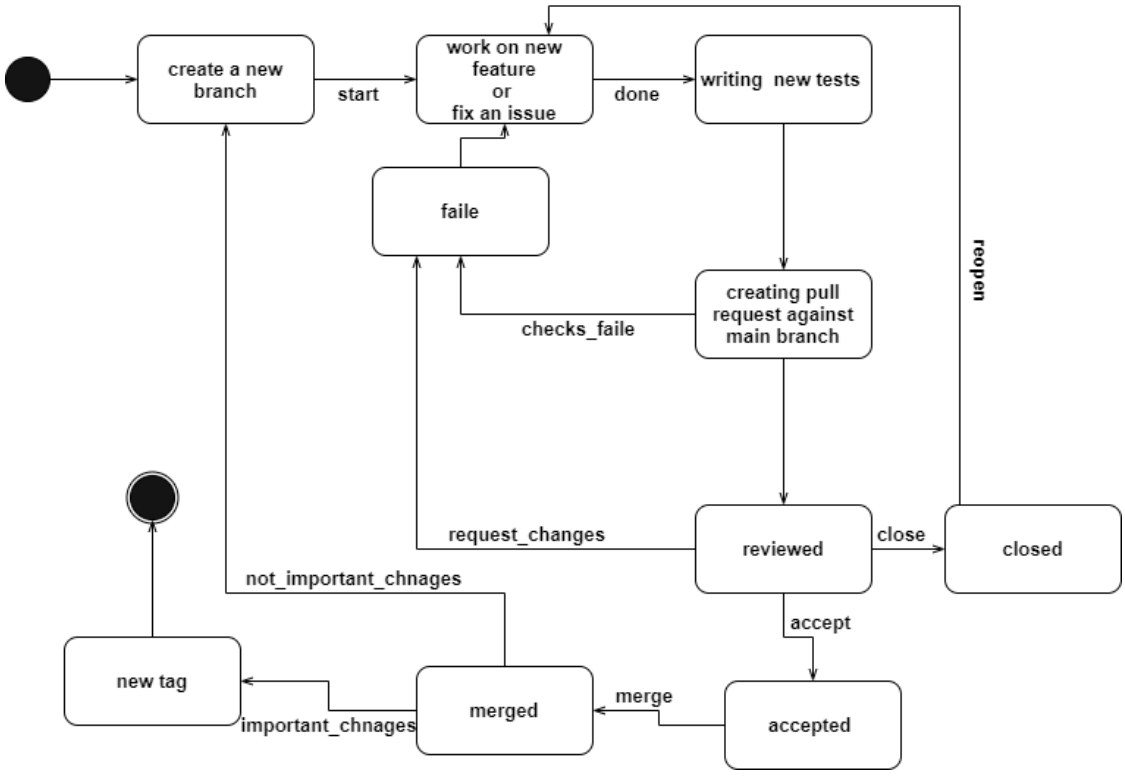

Figure 3.4 Pull request state diagram.

9. Finally, the publish phase It's what we conclude with, this is not the end of the API life cycle it is just one lap that may be repeated many times, its here when the team release a product that can at least satisfy some consumer needs, the team must track the API usage in order to refine the monetization strategy, and to know which link or service is used the most.

3.3.4 Repository of API Lifecycle Management :

Our API lifecycle management process are persisted by using model-based repository.

Querying a relevant aspect of a high-level scientific environment is the possibility of running queries on the API lifecycle management process stored in repository.

To this aim, we have developed a user interface based on the same API, the use of declarative interface offers an easy and effective way to discover process model of the COVERING-API repository.

On the other hand, it eases the manipulation of API lifecycle management process files expressed in our design language (called meta-model) conforms to MOF (meta-object facility) [5], the user can interact with all components via an interface it is a way which is needed to facilitate querying see figure 3.5.

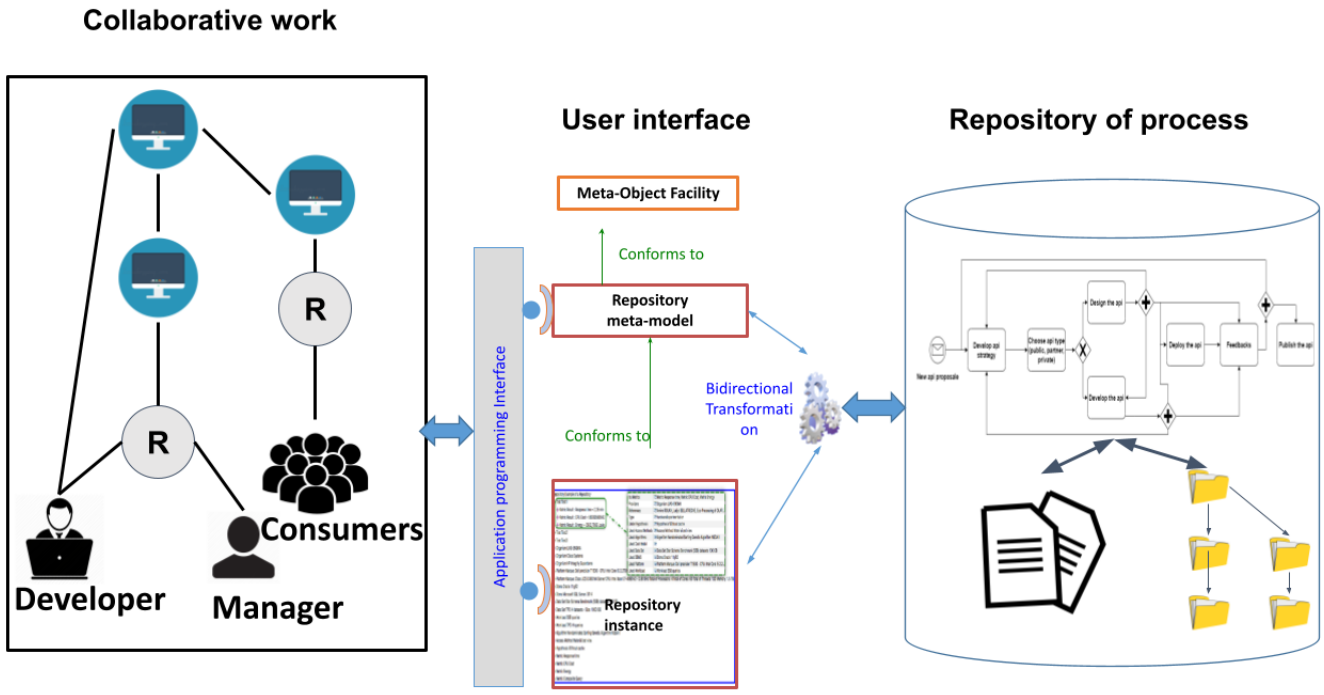

Figure 3.5 Business Process Model Repository of API Lifecycle Management.

3.4 Conclusion

To sum up the results of this chapter we began with a series of issues, that we faced while working and others that we discovered when exploring the literature, and then we extracted the needs from them.

And we have implemented these needs in the form of solutions, which we will support in the following chapter with a tool that will assist those who wish to use our framework while constructing a web API 4.22).

CHAPTER 4. PROOF OF CONCEPT

This is the beginning of the implementation and evaluation phase which focuses on the realization of the conceptual design and our framework as a web application.

We shall justify some technological decisions first and then proceed to illustrate its architecture, then pass and demonstrate the outcome, which will be the first prototype we have produced for the COVERING-API, in the form of figures as well as a link to the prototype on github, and finally we will discuss the evaluation phase.

4.1 Technical Choices

In this section, we will defend our technological decisions, what we based them on, and what considerations influenced our decision to adopt a certain technology over others, we choose based on five parameters, they are discussed in the following list:

- The first of which was to use a data-driven decision-making strategy, as stated by [37] that they were attempting to demonstrate that data-driven decision-making techniques are very relevant and advantageous to Agile software development.

  The combination of data analysis and interpretation by Agile teams of humans can lead to better-informed decisions, in both the business and software development domains, we were startled by the lack or absence of reviews connected to the technologies used in the field in which we operate during our job and our efforts to implement this concept, because of its significance, the decision was made to rely on grayliterature in figure 1.4, which can be found in these sources [15], [43], [1] and [45] this is what the researchers stated about it.

  It was also demonstrated that grey literature can be utilised in software engineering research by [53] and [20], as well as the approach we used in the evaluation, it is discussed in detail here [19].

- The second measure is the proportion of our knowledge with a certain technology.
- Another factor is time, since we had fewer than 12 days to prepare and make the essential decisions to develop this prototype.
- The next aspect we considered was our lack of knowledge in a certain profession, such as user interface design.

  We examine how individuals operate in a specific subject, by gathering a large number of works and then attempt to replicate their work, with the addition of our personal touch.

4.1.1 Web Frontend

4.1.1.1 Programming language

After reviewing surveys conducted by websites, we began to think about and search for what we would pick, which is the core programming language that would govern the code in the frontend of the web application.

We opted to begin our inquiry with Jetbrains since the company is recognized for its credibility, accomplishment, and experience in the domains in which it operates, after reviewing the survey, titled The State of the Developer Ecosystem 2020 [6] it got 19,696 developers involved And our pick was impacted by the results below :

- Javascript is the most widely utilized programming language in general.
- During the 12 months preceding the poll, 70% of developers used javascript, and those who plan to use it or wish to switch to it permanently made up 4% of the total number of participants.
- Typescript was also mentioned in this evaluation, with 28 percent of users using it and 8 percent wanting to transition to it.
- Dart, on the other hand, was scored worse than javascript and typescript by 9 percent of those who use it and 5 percent of those who desire to use it.

From an enterprise to a coding challenge website, it's a HackEarth, according to a poll titled Behind the Code The 2020 HackerEarth Developer Survey [26].

One of his key objectives was to compare student and professional developers, we were able to keep track of certain data which is, javascript was the most popular frontend language, and developers wanted to learn it, followed by typescript and dart. The same thing we noticed on [50], [21], and [13] supremacy for javascript, followed by typescript, and then dart.

This was a popularity analysis of the language that a huge number of developers wish to use.

This leads us to the conclusion that the more widely used technology is where you will find better documentation and more individuals to assist you, in addition to the availability of many open source projects based on the most widely used programming language, as we discovered in particular on github [21].

This is why we rejected dart, as it is a newcomer to the web frontend field and has a lack of documentation and people who can help us, in addition to our lack of experience in it, and as we know learning a programming language with a framework will take a long time, whereas typescript has the ability to, compile down to a version of javascript that runs on all browsers, as well as being a type inference, which gives some of the benefits of types without actually using them, we chose it and then changed to Javascript because we encountered several technical problems that caused the prototype creation to be delayed, plus we have two years of experience with javascript.

We chose typescript and then changed to javascript because we encountered several technical problems that caused the prototype creation to be delayed, plus we have two years of experience with writing code using javascript, check 4.1 to see more details about this surveys.

Table 4.1 Surveys review results of the frontend languages showing developers or projects using the language (developers or projects want to switch to it)

Surveys	**JavaScript**	**TypeScript**	**Dart**
github	94 % (- %)	-	-
Stack Overflow	67,7 % (- %)	25,4 % (- %)	4.0 % (- %)
JetBrains	70 % (4 %)	28 % (8 %)	9 % (5 %)
CodingGame	65.46 % (- %)	-	2.98 % (- %)

4.1.1.2 Frontend Framework

After deciding on javascript, we had to determine whether to utilize a frontend framework or stick with javascript, we picked a framework to have a clean frontend design, as well as to assure faster work and a better user experience owing to the existence of numerous auxiliary tools.

We choose the frontend framework based on data gathered from the pref track by google labs, Angular2 in 4.1 figure was one of the top frameworks, but we couldn't chose it since it used typescript.

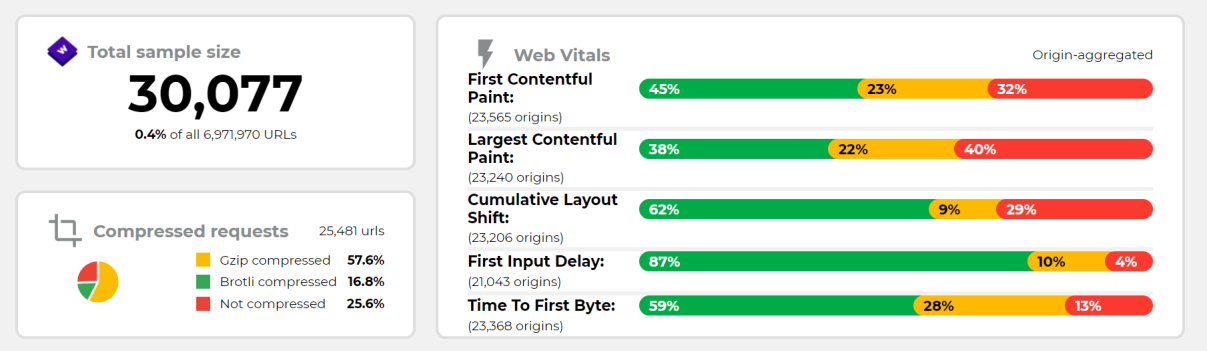

Figure 4.1 Angular2 statistic from Pref Track

The results for Reactjs are shown in 4.2,

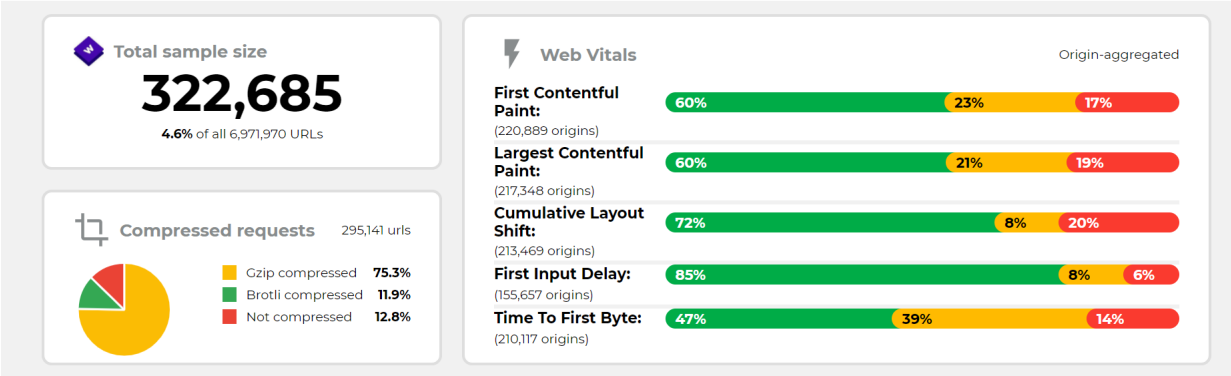

Figure 4.2 React js statistic from Pref Track

along with Svelte in 4.3

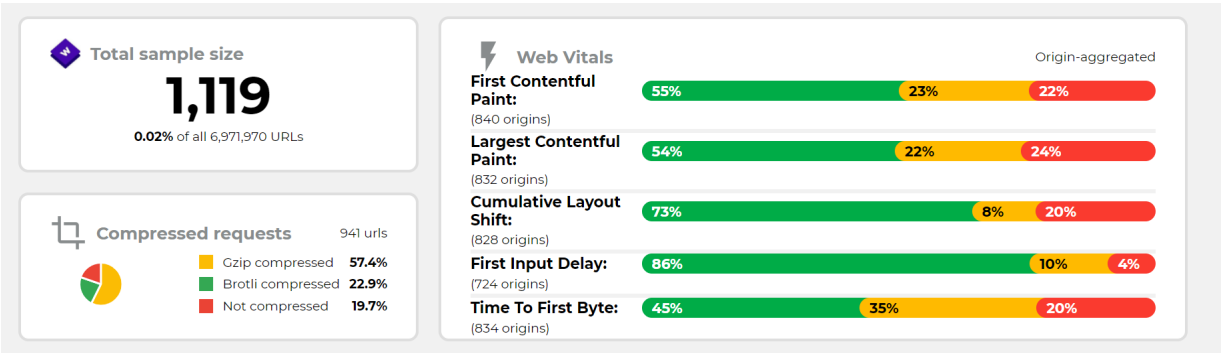

Figure 4.3 Svelte statistic from Pref Track

and Vuejs in 4.4

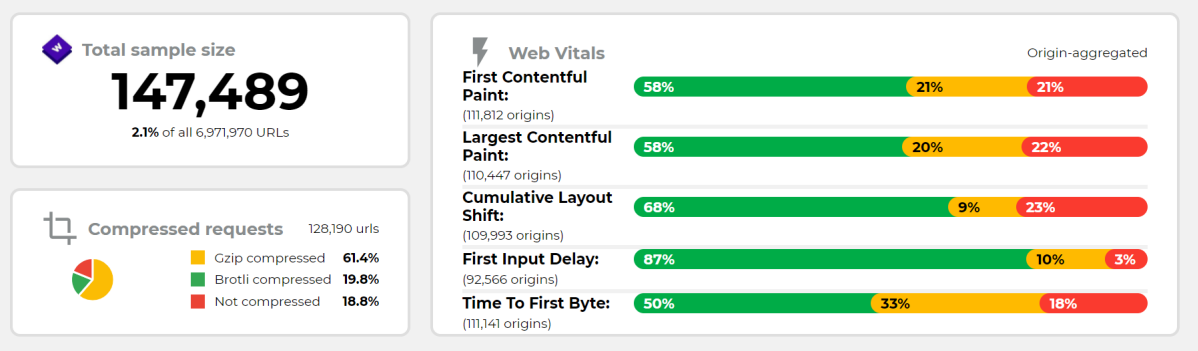

Figure 4.4 VueJs statistic from Pref Track

We chose Reactjs because of the good results and our long experience with it and its frequent use in the web frontend world, as well as the presence of tools that make it easier for us to work based on Reactjs that we used, such as AntDesign and React-router.

4.1.2 Backend

4.1.2.1 Programming Language

In this section of the fourth chapter, we will cover our backend choices and why we chose one technology over another.

So we started talking about picking the right programming language for our system and for ourselves, so we had options such as Java, C sharp, javascript (nodejs), golang, python, php.

We discovered benchmarksgame, a well-known resource for benchmarks in many languages, after a long search for a reputable source that allows us to do a somewhat fair comparison between these programming languages, as far as we are aware.

Regardless of the criticism by both Stefan Marr, Benoit Daloze and Hanspeter Mossenbock in [34] they said that in benchmarksgame requirements for benchmark implementations each implementation must utilize the same algorithm and yield the same results.

Benchmarks for each language can be implemented flexibly beyond these two prerequisites, as a result, the Test Game includes a broad range of implementations in a single

language for the same benchmark, there are alternative ways to represent the method and fundamentally alternative approaches, such as utilizing parallel techniques instead of sequential code, they also stated that, from a scientific standpoint, the Benchmark Game's regulations are overly liberal and allow for an excessive number of participants, there is a considerable deal of variance, which prevents many useful conclusions, based on such standards.

Another issue of debate came from [11] they claim that the benchmarksgame project was developed to assess performance on machines with multiple cores, however the tasks and most of the languages are not suitable for concurrent programming.

According to [12], the benchmarksgame is a good source for comparison applications.

As mentioned in bechmarksgame's how programs are tested webpage, all applications are measured on a quad-core 3.0GHz Intel® i5-3330® with 15.8 GiB of RAM and a 2TB SATA disk drive; using UbuntuTM 21.04 x86 64 GNU/Linux 5.11.0-18-generic.

After reviewing their website and the measures they employ, and despite the criticism raised against them, we decided to rely on them to make our decision regarding the appropriate programming language for us, taking into account our experience with other technologies, as well as the time constraint, which was a major challenge for us during this theses.

We picked the quickest implementation of each of the following algorithms for this purpose, fannkuch-redux, n-body, spectral-norm, mandelbrot, pidigits, and regex-redux written in the programming languages java, c#, nodejs, golang, python, php.

The metrics we used were how much time is taken before forking the child-process and after the child-process exits (sec), and how much memory is consumed (mem),

C sharp was dominant in the initial measure (secs), followed by java, python, and go, which was typically second behind C sharp when we assessed each method independently.

We chose golang since he was the best in memory use (mems), despite our lack of acquaintance with him, our long-term perspective forced us to make this decision, because, in addition to high performance, golang has the ability to endure, we discovered in the subsequent

surveys [50] and [26], that it is one of the most favored programming languages for programmers and they want to learn, particularly in the [6] study.

Go received 18 percent of the vote from the 19,696 people who took part in the poll, and he was first in the proportion of developers who wish to use it as a major programming language or in their projects.

As previously said, our future vision influenced our decisions significantly, we aim to make this a wonderful project that many developers will use and find useful, we also want to attract a big number of contributors to the project's github page.

4.1.3 Other Tools

4.1.3.1 Version Control And Code Collaboration

We want to speak about the most essential tools that we selected to utilize to help us build the *COVERING-API* prototype today, and what may help us grow it into a quality product, so we had many alternatives while attempting to pick the version control system, which required us to go back to the literature.

We started with [33], they identified some limitations with Git one of the most used distributed version control system this issues was as following :

- If someone modifies something in the remote repository, the commit id and history are not immediately displayed on the developer local repository.
- When searching for an unknown author, git terminal produces no results, but he must answer with the statement that the author has not contributed to the project.

They also suggested that these problems might be overcome by adding a logical tree structure to Git to increase its usability.

This tree would contain all of the content as well as metadata and a series of pointers referring to consecutive commits, now we can move to [7] according to their study in a form of a survey of 820 developers, 65 percent use distributed version control systems and 35 percent use

centralized version control systems, in addition to this they mentioned some advantages of distributed version control systems:

- Developers can work in isolation on local copies of the repositories, allowing them to work online while still retaining full project history.
- Developers can create and merge branches cheaply.
- Developers can commit individual changed lines in a file, rather than being forced to commit the entire file as in centralized version control systems.

Due to their lower size, coherent changes, and the inclusion of issue tracking labels, their analysis indicated that distributed version control systems include higher quality commits than centralized version control systems.

They also discovered that distributed version control systems are preferred due to key capabilities such as the opportunity to commit in a local repository, centralized version control systems on the other hand, are recommended for its simplicity of use and a shorter learning process centralized version control systems.

Now we can discuss the finding of [48] they began by comparing centralized version control systems to distributed version control systems and discovered that there are almost no pitfalls to using a centralized version control system over a distributed version control system because the branching and merging features of centralized version controls are already on a equal level with distributed version control systems, this study was quite useful to us since it included a comparison between Git, a distributed version control system licensed under the GPL (GNU) license, and both CVS and SVN, which are centralized version control systems.

Finally we selected Git because we want to benefit from the best choice of enhanced tree management, speed, and functionality by taking 'snapshots' of files will make everything swift and clean in addition to an excellent performance and speed , and that is why we chose to utilize github, a social networking and code collaboration site based on git, to promote, contribute to our project, exchange thoughts, recommendations, and comments about our project. .

4.2 Architecture

We now want to start presenting and discussing the architecture of the *COVERING-API* tool support prototype after we made the technical decisions, which consisted of the tools and technologies that we planned to use, knowing that this is just a prototype and represents only a very simple embodiment of an initial idea for the final product that we are planing to create.

To implement COVERING-API as a design decision, we favored single-page web applications using ReactJs mentioned on 4.1.1, backed by Golang on the server side 4.1.2 and Git as a version control you can see why in 4.1.3 system , figure 4.5 shows an overview about the underlying architecture of the prototypical solution in form of a component diagram.

Our approach has been implemented and made available as an open-source code on github.

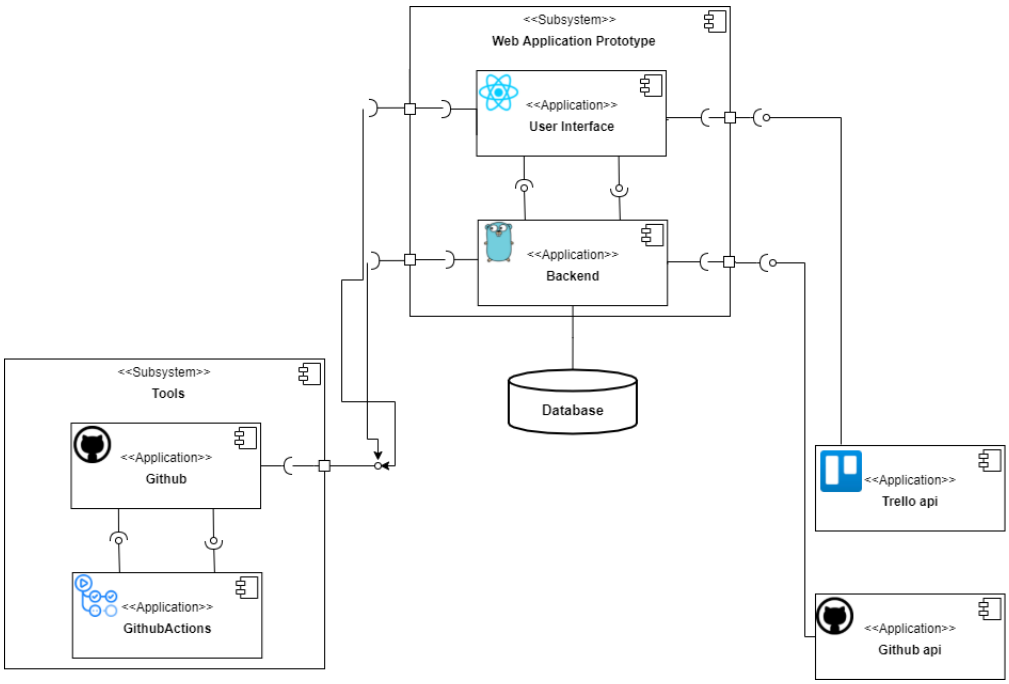

Figure 4.5 Technical Implementation of our Tool Support

We can explain more about our component diagram as mentioned on [32] components, interfaces (typically illustrated in the lollipop notation), and dependencies, which are directed

relationships between two of the other parts, are the three elements of a UML component diagram.

First our system is composed of a two main subsystems which is represented in: *COVERING-API* assistance tool application which comprises two components, one on the client side and one on the server side, and which represents the location of most operations and computations, as well as a connection to the database.

This server transmits the request when requested by the client over the http protocol and is supported by a restful architecture style, the data may be used to get, put, post, and delete data kinds, which refers to the reading, updating, creating, and deleting of resource-related operations, keep in mind that our restful API only supports GET requests, whereas our client, which is built with react js, will send requests to the server side and follow the provision of interactions between the user and the system , figure4.6 show the a simple interaction that happen between the client side and the server side of our tool ,

Out of this subsystems a two dependency represent the concept, which is the dependence of the deployment of the application subsystem on github and his actions, using a continuous integration and continuous deployment pipelines which is one of the fundamental principles of devops this is mentioned on [29], in [52] they said that continuous integration and continuous deployment is one of the practices in devops not a principles they just exemplify the principle of automation and sharing, which communicate by providing interfaces between each of them.

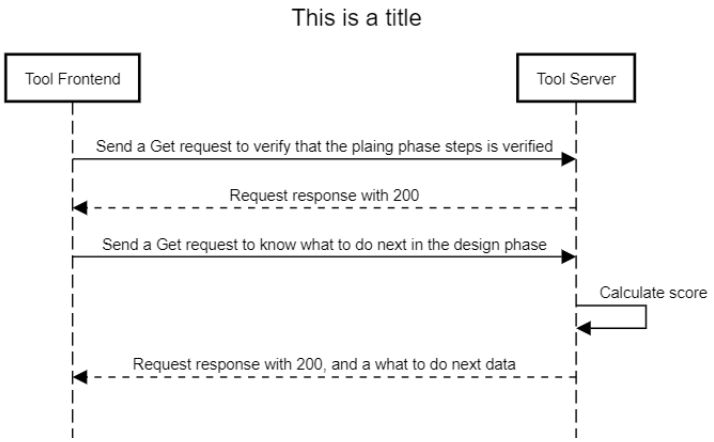

Figure 4.6 A simple Interaction between the client and the server

Another component in this diagram that we must discuss is the github rest API and trello rest API, both of them provide an interface for our system, the first of which is the github web API, which we use to track if a repository, versioning, and the code is adhering to specific metrics that will facilitate collaboration between the API development team check figure 4.7.

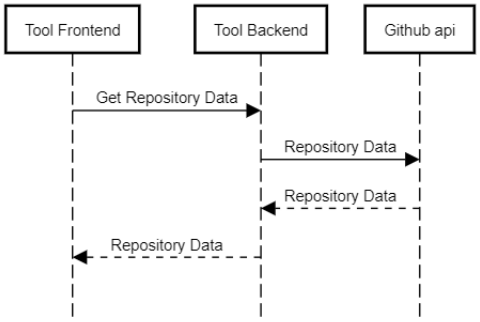

Figure 4.7 A simple Interaction between the client , server and github API

We utilize trello API to track what is occurring in the task board given for the team, as well as to offer a log file to let an API product manager or team leader to follow what his team is doing; we discovered that this functionality is absent in many project management platforms check figure 4.8.

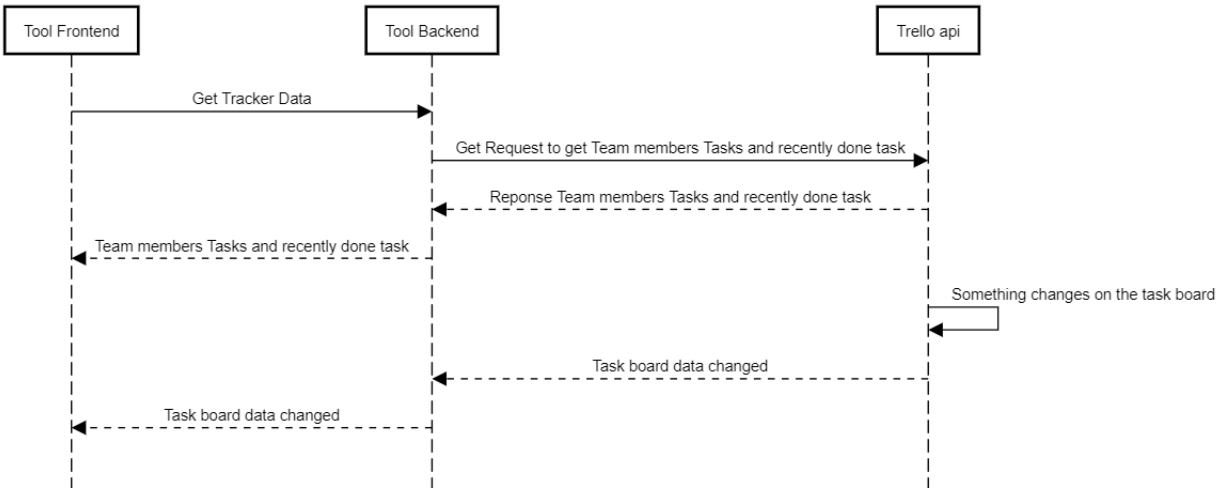

Figure 4.8 A simple Interaction between the client , server and Trello API

An important aspect that we need to discuss is that our architecture is based on the client server model [42], the following are the benefits of a client-server architecture:

- It distributes application processing over many computers.
- It enables simpler resource sharing from client to server.
- It lowers data duplication by keeping data on each server rather than the client.
- It improves integration of business information and system accessibility; and it cuts expenses [14].

4.3 Prototypical Implementation

The COVERING-API prototype help users in the planing, design and phase in for both the technical and non technical choices.

Here also our web application or web app which is a client server software, runs in a multiple browsers and exploits css positioning, with dynamicity that javascript can offer if it is used in every single software on the web, customisation and capabilities of automatic arrangement using flexible box layout (Flexbox) with the aid of ant material design.

We will now introduce the various features of our tool and the development support provided by the system to ease the understanding.

4.3.1 User Interface

As we discussed in on of the previous sections 4.1, we used a technique that focuses on collecting as many user interfaces designs as possible for web applications similar to what we do, except that it is a user interface design for applications centered around areas we have identified, and it includes assistant tools, administration, and statistics dashboards, knowing that we collected them from sites such as behance, dribble and muzli.

These user interfaces all contain a sidebar with some items on it, usually the position of this sidebar is on the left side of the screen, these items are accompanied with a word or two next to an icon to provide the user with a clear understanding of the function of this sidebar item and this is shown in the figure 4.9.

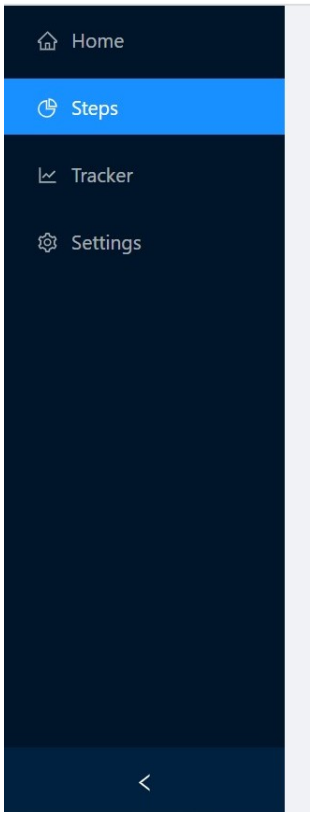

Figure 4.9 *COVERING-API* Tool Support Dashboard Sidebar

The sidebar elements must also be able to alter form, background color, or text as the cursor passes over them, which is known as hover effects, a look may be made on the figure 4.10, which demonstrates this, and there is also an active effect, which must alter the background of the navigation bar item when it is finished.

When you click on it, the selection will be completed as shown in the figure 4.9, and this can be observed by noting the steps item, its color #1890FF differs from the navigation bar background color, which is #001529.

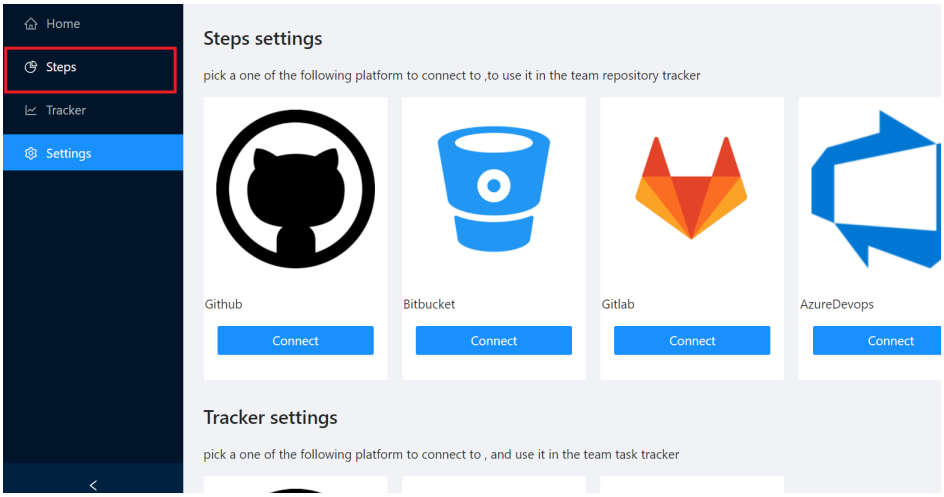

Figure 4.10 *COVERING-API* Tool Support Show Dashboard Sidebar Item On Hover

According to what we have read in both [56] and [36], we must emphasize that the text must always be clear in order to provide a pleasant user experience, a contrast ratio of at least 4.5:1 is required for the visual display of text and text pictures.

In addition to the sidebar, we stated that what we gathered from the user interfaces is distinguished by the use of fewer colors, as well as the fact that when graphs or charts are used, the colors are different from the colors used in the rest of the application to ensure their clarity because they represent an important source of information for the user.

4.3.2 Design and Planing phase

The sidebar's function in this case is to allow the user to move between the various pages in the application, when you first open the application and click on the step sidebar item, you will see the user interface for the planning phase that we discussed before.

As indicated in the *COVERING-API* framework guidelines, this interface provides a series or phases that the user must perform until the planning stage is done check figure 4.11.

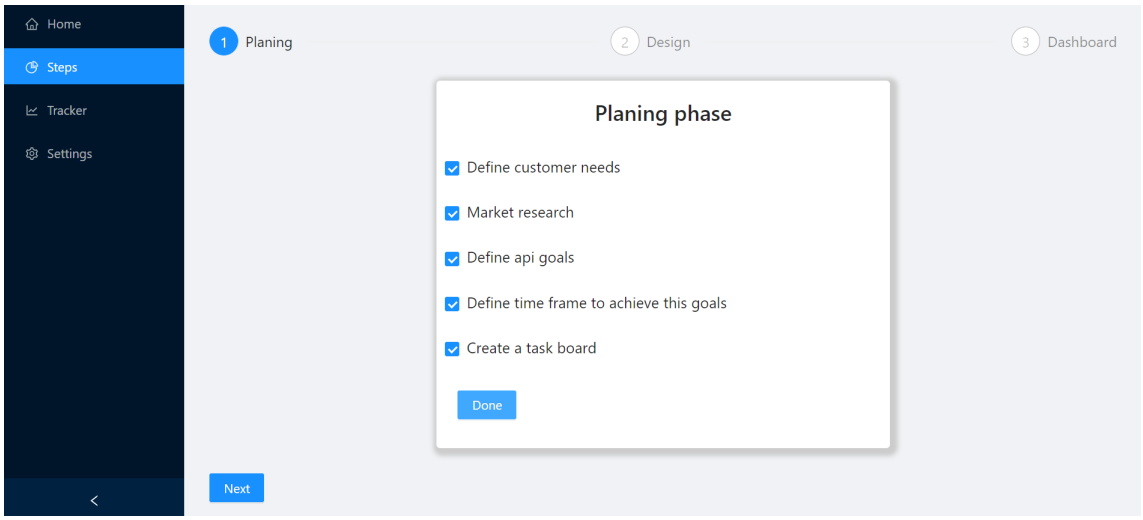

Figure 4.11 Show Dashboard Of *COVERING-API* Tool Support Showing The Planing Phase

When he hits the next button, he will be sent to the interface shown in figure 4.12, where a series of questions will be presented to the user in order for the system to advise what to do next in the design process.

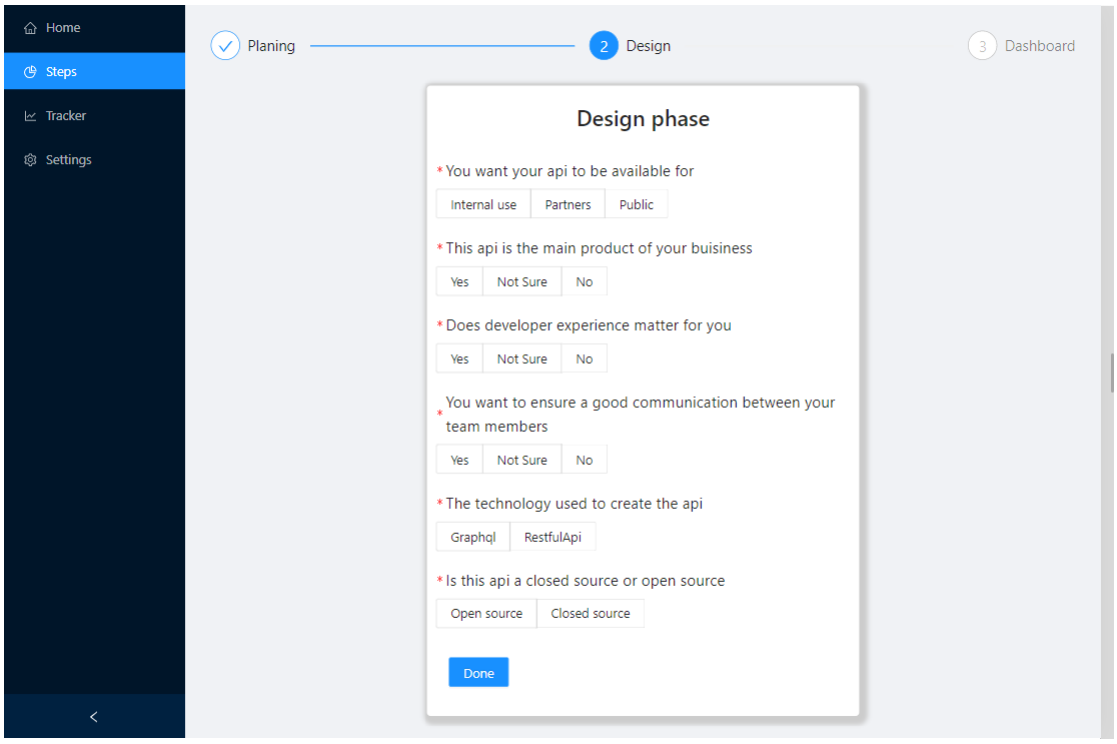

Figure 4.12 Show Dashboard Of *COVERING-API* Tool Support Showing The Questions Part of Design Phase

When these questions are answered, the user will be asked whether he wants to go to the coding phase 4.13 or continue with the design phase, and then a series of steps will be displayed in the form of questions 4.14 that the user will answer, and when he is finished, he will press the next button to proceed to the next stage, which is the same stage he would go to if the coding phase was proposed.

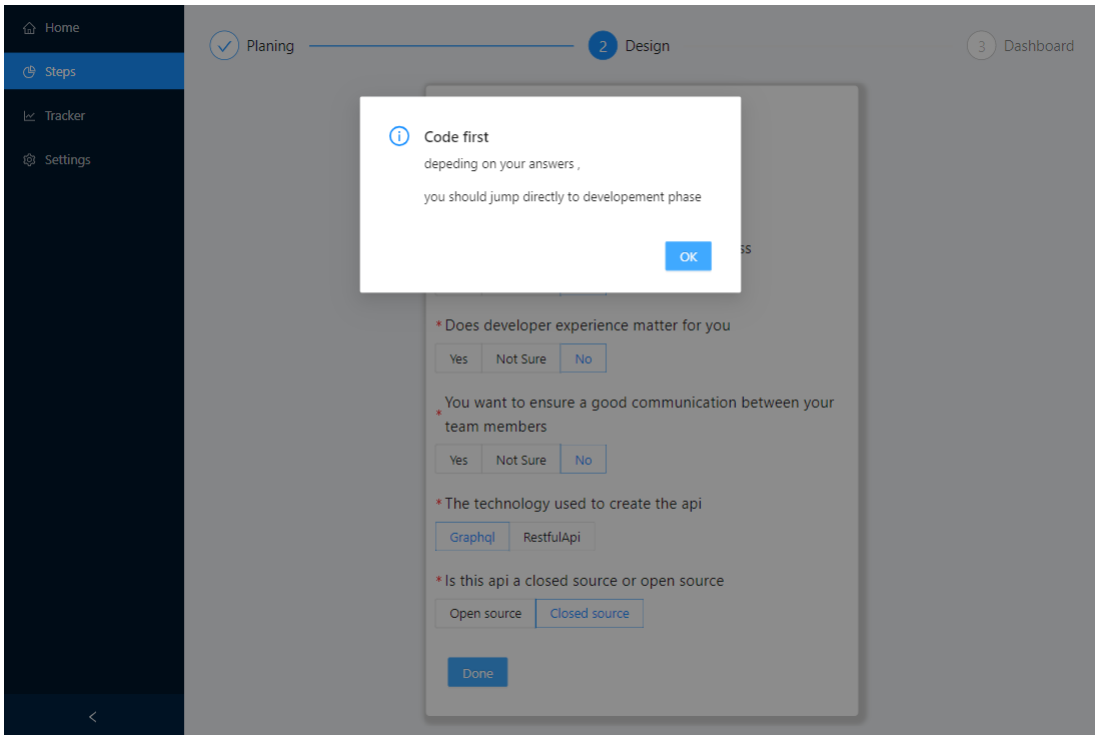

Figure 4.13 Show Dashboard Of *COVERING-API* Tool Support Showing The System Suggest To Start with Coding

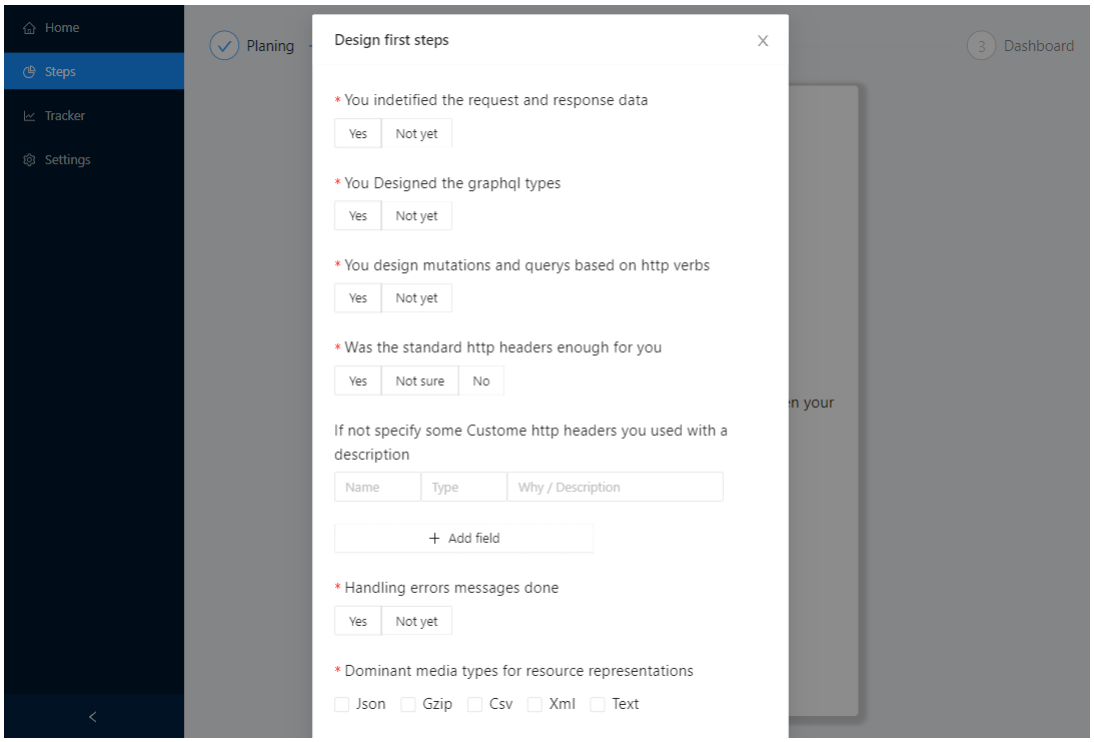

Figure 4.14 Show Dashboard Of *COVERING-API* Tool Support Showing The System Suggest To Start with Design First

4.3.3 Repository Checker

Finally the user can see a dashboard he will then choose his API project repository on github, our system will analyze the repository by connecting to the github API, the user can see the number of commits, contributors, pull requests, issues, merges against the main branch, and a bar chart showing the the open, closed issues and pull requests, on the left side there is a card show if the main branch of this repository is restricted and how many reviews is required to merge to this branch, in the bottom of the page our user can see two pie charts showing the number of commits and branch names that respects the well-known commit and branch naming standard, in the left he will see if the tags or the releases is respecting the semantic versioning or date based versioning standards , see figure 4.15

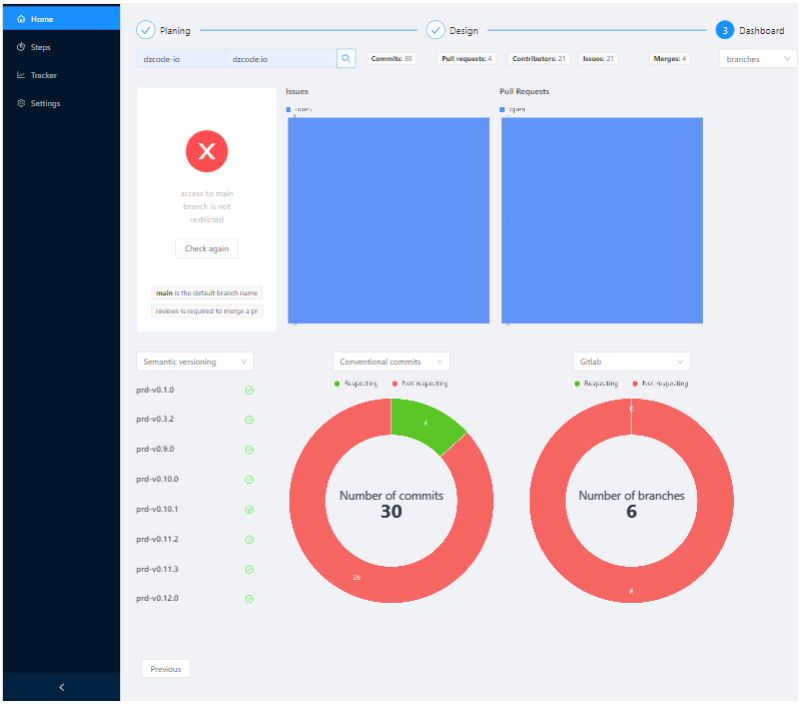

Figure 4.15 Show Dashboard Of *COVERING-API* Tool Support Showing The Code Repository Checker

4.3.4 Task Board Tracker

If the user click on the third item in the left side bar menu our trello task tracker will appear on the page, the task tracker have log of everything happening for example if a member of the team rename a task, add new items to the task or add comment our tracker will show all of this and more, the user also can set a time to each task in the Trello board our tool will track it and the result will be delivered to him when time is up, on the left bottom side of the tracker page the user can see the last finished tasks by the development team , see figure 4.16

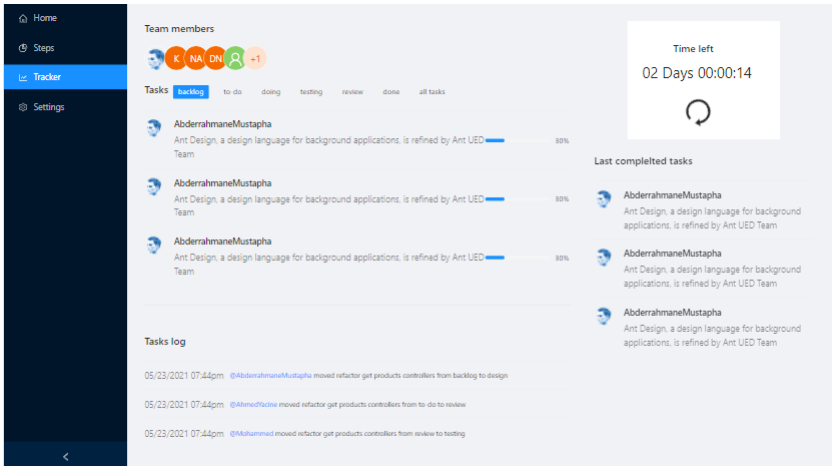

Figure 4.16 Show Dashboard Of *COVERING-API* Tool Support Showing The Task Board Tracker

4.3.5 Backend

In terms of the backend, we discussed utilizing golang to connect to a basic database, which is the json file, save the processes, validate them, and then deliver them to the client side in the form of json responses through rest full architecture based on http protocol, check figure 4.17.

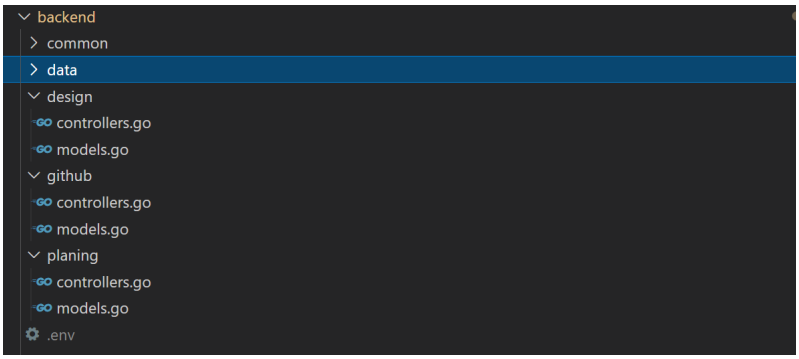

Figure 4.17 Show The *COVERING-API* Tool Support Backend File Structure

Another role is that the backend communicates with the API's, which in turn sends a response containing the object it belongs to, and is structured according to the API, the backend

converts it into its own object and sends it to user interfaces such as task board tracker and repository checker.

As an example, consider what we did with the github API, especially with the pull requests, which we get from the following url, and the data is given to us, as seen in the figure 4.18 below.

Figure 4.18 Show The Response of a GET Request To This Url From github API

We didn't need all of this information, so we used the struct shown in figure 4.19 to pick only the fields we needed and to modify the names of some of the fields if we wanted to, in goLang, struct is comparable to Object or Class in languages such as Java and C.

Figure 4.19 Show Struct That We Used To Limit The Fields That We Fetched From The Pull Requests Url

As for how to convert this github object into a covering API tool object, it is shown in a code in figure 4.20

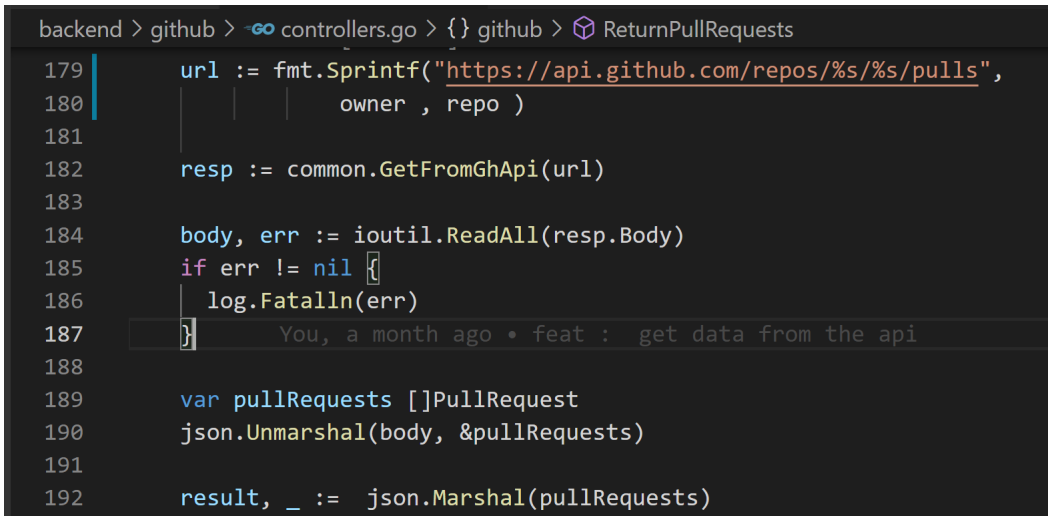

Figure 4.20 Show How We Converted github API Object to *COVERING-API* Tool Object

The code for converting this github item into a *COVERING-API* tool object is illustrated in figure 4.20, in the line 179 we are sending a get request to this a url mentioned in the github API documentation, the response was a list of pull requests, we converted this pull request from a github pull request object to a *COVERING-API* tool object in the line 190.

4.4 Evaluation

Following the previously described details of the prototypical *COVERING-API* implementation, the chapter basically deals with the evaluation of the prototype and *COVERING-API* framework to validate assumptions, gain feedback on the current status of the realized framework and the prototypical solution, the objective of the review is to establish whether the general idea of our solutions is appropriate for a small team in a business environment.

The evaluation, in particular, offers an understanding of the supporting features for the API lifecycle and evaluates if the chosen features are useful in fulfilling the requirements described here.

And we'd like to point out that, due to time constraints, we were only able to get feedback from two senior web full stack developers, two junior Frontend developers, and three others who are still studying as software engineers.

Knowing that we only did interviews with the senior developers, we took their opinions about the tool, and the questions in the interview were as follows.

- What difficulties do you usually encounter when trying to develop an API?
- When developing an API, did you try to divide the process into multiple stages? If yes, please tell us?
- Why choose API first or design first strategy ?
- As for the tool, how do you think it can help you to create a web API

- What are the problems in this tool that you see?
- What do you want to see in the future releases of our tool?

The findings showed that this tool is a solid idea, but it has certain flaws that are not found in a comparable tool since it is easy to use and basic.

This was both a strength and a weakness, because our solution solved issues for a small team or a single developer, and it was also one of the most important outcomes of this project.

We need to work on it further and perform more iterations to solve the issues and provide additional features that help with API lifespan and cooperation management, as well as a code generation tool.

4.5 Conclusion

We have completed our work on the prototype, which is simply a simple representation of our idea, and we have demonstrated that it is feasible to transform *COVERING-API* into a tool that assists developers in implementing this framework, and, due to a lack of time, we conducted a short evaluation stage, knowing that we would continue to work on this tool and plan to work on validation and publish a scientific paper.

PROJECT MANAGEMENT

Prior to commencing the project, we had to pick the approach for project management that would fit us for the whole duration of this theses, during this chapter we will also present the development strategy employed during the implementation of the *COVERING-API* framework.

4.6 Methodology

To carry out an IT project there are various development techniques, the chosen approach must fit well to the circumstances of the project, this theses demands all the collaborators and stakeholders to be continuously involved, we didn't have a clearly defined objectives we had just an overview about the final goal which was to define workflow for a small team or business if we had the opportunity to do so, to help them achieve a specific goal during the creating of an API.

After working with stock and buy we couldn't supply the product as we anticipated it, we then moved to the theoretical section to solve many of the difficulties, where we sought to address the problems we experienced while we worked on the prior project, that we could not find obvious solutions for in the academic or industrial areas, in response to the changes we expected to confront and our lack of clear goal setting, we picked the agile methodologies and the toyota technique, Kanban, which began more than 50 years ago.

Kanban is an occupational management technique that helps you visualize and restrict the work that's underway.

It may frequently serve as a strong tool for teams who wish to make their work environment more transparent, communicated and harmonized, Kanban is based on a task board, the process of building and maintaining a Kanban board, which is a project management tool that employs cards and columns to compose a visual "workflow.", figure 4.21 shows an example of a Kanban board.

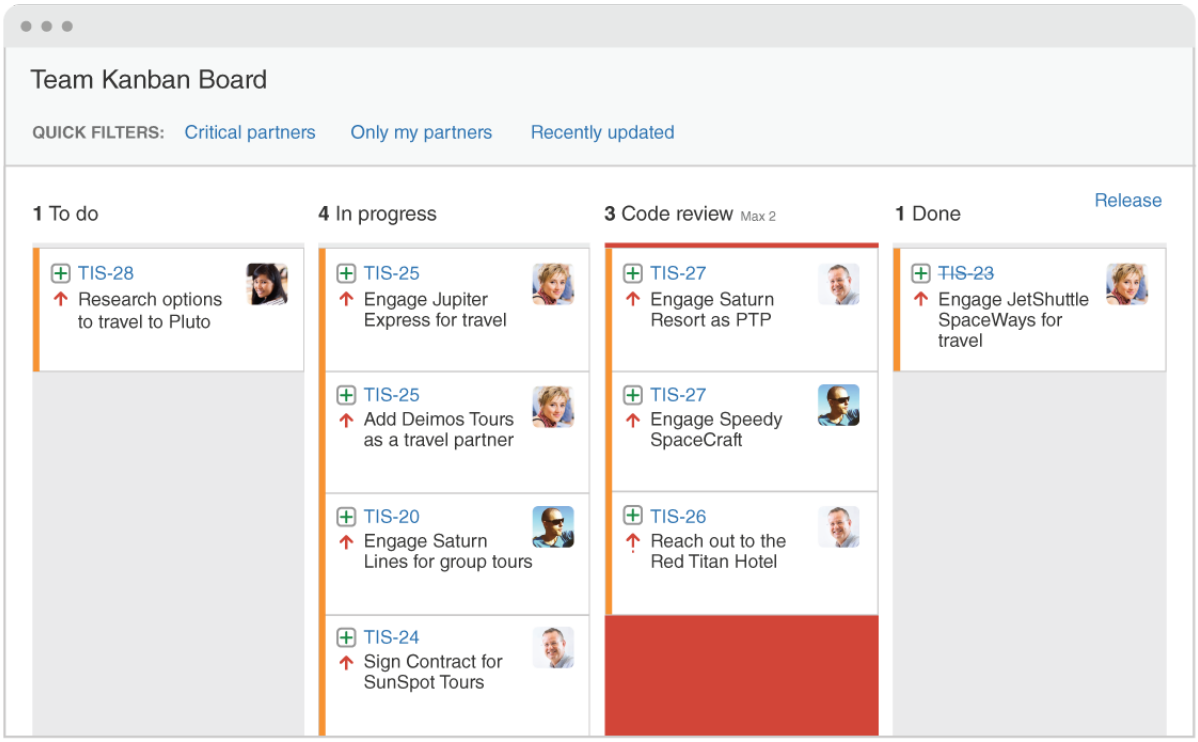

Figure 4.21 Kanban Task Board Example From Atlasian

4.7 Development Process

4.8 Internship

On January 10 we started our internship with StockAndBuy on a meeting with Mr Taher the owner of this company, my colleague Ali Kouriba, and Mr Fethi Boukhors who deserves a lot of credit for introducing us to Mr. Taher this was just a meeting to know more about each other Table 4.8 shows more about this meeting.

When we weren't meeting, we utilized the trello task board (shown in figure 4.22), as well as Slack, which was really helpful; you may try it in your next project to learn more.

Participants	Date	Goals
Taher Ouhrouche, Fethi Boukhors, Ali Kouriba, Abderrahmane Toumi	Sun Jan 10, 2021 from 2pm to 4.30pm	To know each other more.
Taher Ouhrouche, Abdelkader Ouared, Ali Kouriba, Abderrahmane Toumi	Sun Jan 17, 2021 from 5pm to 7pm	Was named as master internship kick off, the goals was to present the academic expectation and the exacution plan about how we can create the APIs Integration.
Taher Ouhrouche, Abdelkader Ouared, Ali Kouriba, Abderrahmane Toumi	Sun Jan 31, 2021 from 4pm to 6:30pm	Our goal here is to present our progress on the theory part and the stat of the art Stock&Buy introduction and how it is built here we changed our objective from building integration of Amazon and Shopify to creating Web API for stock and buy.
Taher Ouhrouche, Abdelkader Ouared, Ali Kouriba, Abderrahmane Toumi	Mon Feb 1, 2021 from 5pm to 6pm	To walk through the architecture and plans for the next couple of weeks, also to tell us what we should learn more.
Taher Ouhrouche, Abderrahmane Toumi	Sun Mar 21, 2021 from9:30am to 10:30am	to see our progress in developing the API, in this meet Mr Taher told me to start witting unit tests.

Table 4.2 Main Internship And Theses Meetings

After this a meeting was planned with Mr Ouared and Mr Taher to set the goal of our internship. In the beginning the goal was to integrate the Amazon, Shopify APIs that StockAndBuy need to deliver a better services for his customers, we did a state of the art and searched about the recent work in the industry part, it took us 15 days to complete this, in the

second meeting the plan has changed, based on customer needs and that there is not alot of researches opportunity in the field of APIs integration.

So we switched into developing an API product for StockAndBuy to make it available for consumers to make application to based on the data and the services that StockAndBuy offer.

In collaboration with Mr Ouared, Mr Taher started a meeting to explain more about StockAndBuy and the objectived of our internship, in the other side Mr Ouared presented the academic objectives of the internship and the theses, here begins the role of kanban, where we created the task board to create the Web API, which we assumed will be ready by the end of May, figure 4.22) shows us StockAndBuy API Task Board.

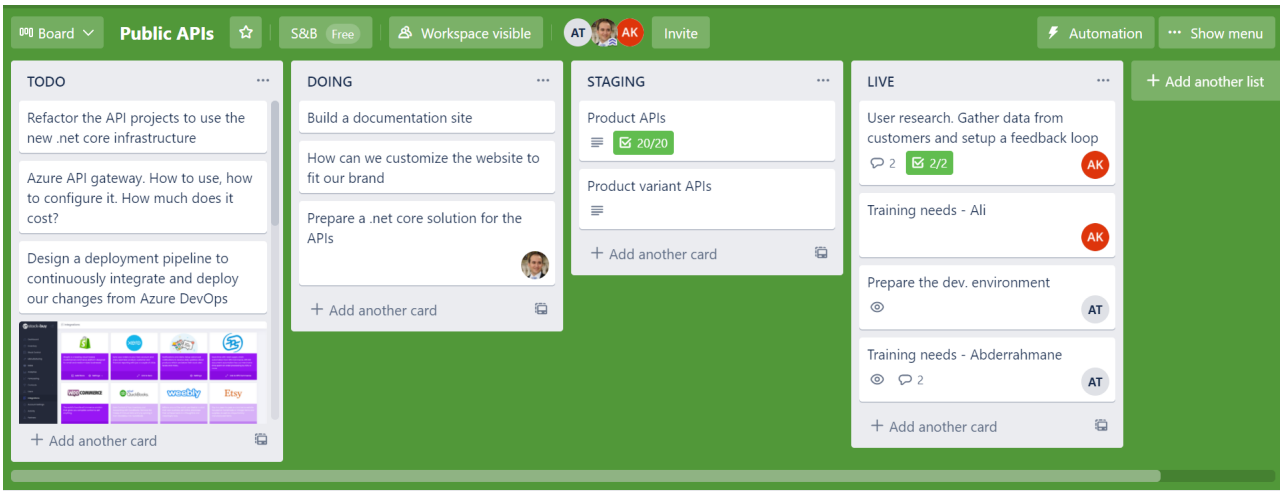

Figure 4.22 Kanban Task Board Of StockAndBuy API Developement

In the end the wind did not blow in the direction that the ships desired, we only delivered the fist part of the StockAndBuy API which was about products and variants API with its documentation on May 26, this was because of a lot of issues that we faced during our internship

- First of all the goals was not clear and we did not had a lot of experience with the best practices of developing API, and the technologies used in the development of StockAndBuy

API it took us a one month of continues training to adapt to this technologies including C#, Microsoft Azure cloud platform, azure functions.

- Another issue that impacted us the most was that during the API development process in the 01 March, i finished most of the parts of the first release of StockAndBuy, API but Mr Taher was hit by an unexpected surprise for one month so we did not hear anything from him, and at the same time Mr Ouared was too busy dealing with some other unexpected conditions, so we did nothing for on,

- After that exams started, the exams took longer than anticipated, the strike, which was initiated by the student unions, lasted two weeks.

  We didn't know when to return because no date for the end of the strike had been established for us, so we were late.

I delivered the fist part of the StockAndBuy API on 26 May with its documentation that you can check it here

4.9 Theses

It was here that Mr Taher suggested on April 20 that we shift our focus to the academic and theoretical aspects of our theses because we didn't have a lot of time, so we created our Kanban task board, which is shown in figure 4.23.

As we mentioned in section 4.8, an overview of our goals in this theses was set, but at this point in time our goals were clear, we truly know what we should do; it's a methodology and framework that will make API design and development easier and more transparent for small teams and developers.

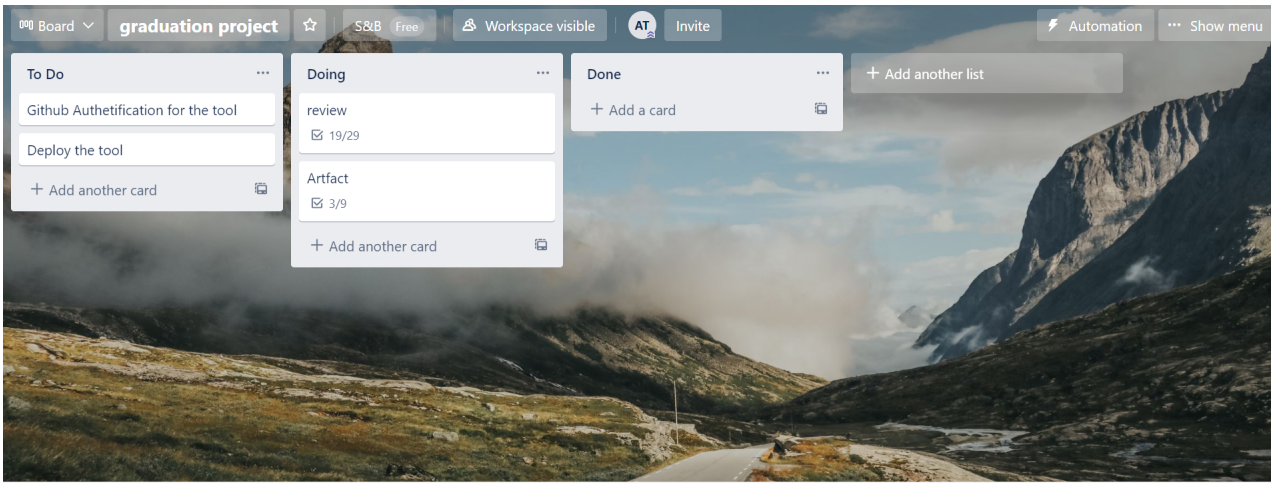

Figure 4.23 Kanban Task Board Of Our Theses

We wanted to solve some of the problems that we faced during our internship which is mentioned in this section 3.1, so we successfully created the first version of *COVERING-API* with a prototype of its tool support, and we evaluated it by meeting with an expert and another junior back-end developer, and Vansath the owner of fireAPIs illustrated in 4.3, and we are planing to do a polling the other candidates based on questions from our survey all of this candidates are a students in the university,

Our aim here was to investigate how our tool may make API development easier for someone who is unfamiliar with this topic.

Interviewee	Type	Goal	Results
Zakaria Mansouri (full stack web developer)	interview	Usefulness of the prototype and the impact in the open source projects.	This tool can be so helpful in the process of creating web APIs and showing the developers what do next, its also have a high potential as an open source tool but it need more work because its only solve simple problems, another suggestion was the ability for a developer to get benefits of this tool directly from a read me file as badge
Haroune Mohammedi (back-end developer)	Interview	Usefulness of the prototype and how it can help you as a developer	developers need to guide and assist them through the process of designing API link
Vansath (owner of fireAPIs)	Messaging	Usefulness of the prototype and what is the challenges that developers face during the creation of API	his view point of API life cycle is that developers will have some blocker or hassles in designing API structure(such as how many tables he needs to retain or which one he needs to keep for which primary key)

Table 4.3 Interviews meetings

4.10 Conclusion

This how we managed our project and delivered a framework to support developers during the journey of developing API's, plus and additional prototypical implementation in a form of web application in order to make the process easier for those who want to create Web API's using our framework.

.

CONCLUSION

This thesis explored, designed, and developed a framework to manage the development and creation of web APIs throughout their lifecycle, we also discussed and evaluated the implementation of a support tool for this framework.

Throughout this journey, we followed the design science framework by [27], who stated that building an artifact in design science research requires many iterations, we also followed [54] article, which taught us how to construct theories before beginning the design science process, the utility hypotheses we imposed were as follows if our framework *COVERING-API* is used correctly, developer know what to do next so he can deliver an API faster and a team can collaborate better during the development.

And as is typical in research, we proposed two study questions, which are reflected in,

Reasearch Question 01 : How can we provide a framework and a set of successive phases and guidelines to demonstrate the path of an API lifecycle for a small team or a developer? Our aim was to create a framework called *COVERING-API* which is based on its core components and the different links between the different stages in a given project to create a complete and a use full API product that the consumer can benefit from its usage.

Another goal, when we asked this question, which has motivated this choice we have proposed a design languages for *COVERING-API* as an intermediate framework between API developers and API product manager, this framework is based on the *COVERING-API* language which is conceived as a DSL (domain-specific language) for *COVERING-API* management system language is composed of a set of packages.

we also attempted to begin an effort to establish an uniform perspective of the API lifecycle with this question.

Reasearch Question 02 : How can we implement this framework as a tool to help a small team or developer create a successful weB API product?

In order to answer this question we planed, designed and then implemented a web application prototype, to help the developer or small team and guide him as well as what he should do in this stages, in addition to helping him and suggesting the best thing to do as a next step, depending on the *COVERING-API* framework?

This tool should be able to analyze the main repository in which the team works and encouraging the user to follow metrics, and guidelines in order to ensure better collaboration for the team and more inclusion for one developer, it should provide a simple way for the team leader to follow what other members are doing during the course of the project.

Our results was that this tool can be so helpful in the process of creating web API's, and showing the developers what do next, its also have a high potential as an open source tool but it need more work because its only solve simple problems.

The tool support for our framework have the task board log feature which gonna help the team leaders or API product manager to know more about what the team is doing, another suggestion was the ability for a developer to get the benefits of this tool directly from a read me file as badge, also there is some missing features first we can talk about that developers need to be guided and assist them through the process of designing API links.

Also during designing API and developing it developers will face challenges such as how many tables he needs to retain or which one he needs to keep for which primary key, this is what we discovered through our interviews and observations.

It is evident that this was only the first stage in a lengthy process that will require extensive reform and refinement in subsequent versions, although we only completed one tiny iteration because we didn't have enough time, also, in order to define what we want to accomplish in order to publish a scientific article, we want to develop a unified picture of the API lifecycle and collaborate during it.

4.11 Future Work

This work opens several directions of further research. From theoretical point of view, we are: (i) Add a feature that helps to develop more secure api, (ii) A better assistant in the planing phase (iii) incorporating the code generation in the process of API life cycle such as documentation and rapid prototyping (iiii) A graphic user interface to create API specification using openAPI and generate code from it.

From practical perspective, we are conducting other experiments with very large teams and data set to evaluate validate the effectiveness and efficiency of our framework, one of this experiments is that we are trying to create a public web API about algerian wilaya or provinces in collaboration with Dzcode community .

From methodological point of view, we are also planning to investigate the possibility of integrating the developer experience design dimension in earlier API conception phases and trying to check if varying analyze stockholder needs.

4.12 Personal Appreciation

During this time working on these theses, i learnt a lot, the most essential of which is how to draw lessons from failure and apply these lessons to solve our issues and benefits from them. I also learnt how to do project management and how i can work with a team, why should we establish our goals and the outcomes we want to achieve in each project or step we want to take in our life, as well as how to adjust to difficulties that arise unexpectedly and that we had not prepared for in advance.

In addition, there were many wonderful individuals from whom i learnt a great deal.

The most essential thing i learnt was how to connect with people more effectively, as well as how to write in an academic setting, such as theses or scientific articles, the internship with StockAndBuy had a significant impact on my learning, more information about cloud computing and microservices.

Furthermore, developing an API is more than just writing code and providing links to other developers; it is a stand-alone product with its own lifetime, and i am grateful to have had this wonderful experience.

BIBLIOGRAPHY

[1] Adams, R. J., Smart, P., and Huff, A. S. Shades of grey: guidelines for working with the grey literature in systematic reviews for management and organizational studies. *International Journal of Management Reviews 19*, 4 (2017), 432–454.

[2] API, O. Open api official documentation.

[3] Awasthy, R., Flint, S., Sankarnarayana, R., and Jones, R. L. A framework to improve university–industry collaboration. *Journal of Industry-University Collaboration* (2020).

[4] Axway. Full lifecycle api management.

[5] Bork, D., and Fill, H.-G. Formal aspects of enterprise modeling methods: a comparison framework. In *2014 47th Hawaii International Conference on System Sciences* (2014), IEEE, pp. 3400–3409.

[6] Brains, J. The state of developer ecosystem 2020, 2020.

[7] Brindescu, C., Codoban, M., Shmarkatiuk, S., and Dig, D. How do centralized and distributed version control systems impact software changes? In *Proceedings of the 36th International Conference on Software Engineering* (2014), pp. 322–333.

[8] Bucena, I., and Kirikova, M. Simplifying the devops adoption process. In *BIR Workshops* (2017), pp. 1–15.

[9] Bui, D. H. Design and evaluation of a collaborative approach for api lifecycle management. TECHNICAL UNIVERSITY OF MUNICH.

[10] Cadavid, J. J., Combemale, B., and Baudry, B. An analysis of metamodeling practices for mof and ocl. *Computer Languages, Systems & Structures 41* (2015), 42–65.

[11] Cardoso, R. C., Hübner, J. F., and Bordini, R. H. Benchmarking communication in actor-and agent-based languages. In *International Workshop on Engineering Multi-Agent Systems* (2013), Springer, pp. 58–77.

[12] Castro, D., Hu, R., Jongmans, S.-S., Ng, N., and Yoshida, N. Distributed programming using role-parametric session types in go: statically-typed endpoint apis for dynamically-instantiated communication structures. *Proceedings of the ACM on Programming Languages 3*, POPL (2019), 1–30.

[13] CodingGame. Codingame 2020 developer survey report, 2020.

[14] Duchessi, P., and Chengalur-Smith, I. Client/server benefits, problems, best practices. *Communications of the ACM 41*, 5 (1998), 87–94.

[15] Farace, D., and Schöpfel, J. *Grey literature in library and information studies*. KG Saur, 2010.

[16] Farley, D. Continuous integration and feature branching, Mar 2018.

[17] Farley, D. Continuous integration vs feature branch workflow dave farley goto 202, Jun 2021.

[18] Fielding, R. T. Architectural styles and the design of network-based software architectures.

[19] Garousi, V., Felderer, M., and Mäntylä, M. V. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology 106* (2019), 101–121.

[20] Garousi, V., and Rainer, A. Grey literature versus academic literature in software engineering: A call for epistemological analysis. *IEEE Software* (2020).

[21] Github. The 2020 state of the octo—verse, 2020.

[22] Godin, K., Stapleton, J., Kirkpatrick, S. I., Hanning, R. M., and Leatherdale, S. T. Applying systematic review search methods to the grey literature: a case study examining guidelines for school-based breakfast programs in canada. *Systematic reviews 4*, 1 (2015), 1–10.

[23] Gomez, P. R. *Service Oriented Architecture as a strategy for business improvement in the enterprise*. PhD thesis, Massachusetts Institute of Technology, 2008.

[24] Google. The api product mindset.

[25] Google. The digital transformation journey.

[26] HackerEarth. Behind the code - the 2020 hackerearth developer surve, 2020.

[27] Hevner, A. R., March, S. T., Park, J., and Ram, S. Design science in information systems research. *MIS quarterly* (2004), 75–105.

[28] Jacobson, Daniel, G. B., and Woods, D. Apis: A strategy guide. In *APIs: A Strategy Guide. O'Reilly Media, Inc.* (2012), O'Reilly.

[29] Karamitsos, I., Albarhami, S., and Apostolopoulos, C. Applying devops practices of continuous automation for machine learning. *Information 11*, 7 (2020), 363.

[30] Kopecký, J., Fremantle, P., and Boakes, R. A history and future of web apis. *it-Information Technology 56*, 3 (2014), 90–97.

[31] Lane, K. Intro to apis: History of apis.

[32] Lüer, C., and Rosenblum, D. S. Uml component diagrams and software architecture-experiences from the wren project. In *1st ICSE Workshop on Describing Software Architecture with UML* (2001), Citeseer, pp. 79–82.

[33] Majumdar, R., Jain, R., Barthwal, S., and Choudhary, C. Source code management using version control system. In *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)* (2017), IEEE, pp. 278–281.

[34] Marr, S., Daloze, B., and Mössenböck, H. Cross-language compiler benchmarking: are we fast yet? *ACM SIGPLAN Notices 52*, 2 (2016), 120–131.

[35] Massé, N. Full api lifecycle management: A primer.

[36] materialui. Accessibility. materialui.

[37] Matthies, C., and Hesse, G. Towards using data to inform decisions in agile software development: views of available data. *arXiv preprint arXiv:1907.12959* (2019).

[38] MuleSoft. What is an api.

[39] Nah, F. F.-H. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology 23*, 3 (2004), 153–163.

[40] Nally, M. grpc vs rest: Understanding grpc, openapi and rest and when to use them in api design.

[41] Nybom, K., Smeds, J., and Porres, I. On the impact of mixing responsibilities between devs and ops. In *International Conference on Agile Software Development* (2016), Springer, pp. 131–143.

[42] Oluwatosin, H. S. Client-server model. *IOSRJ Comput. Eng 16*, 1 (2014), 2278–8727.

[43] Paez, A. Gray literature: An important resource in systematic reviews. *Journal of Evidence-Based Medicine 10*, 3 (2017), 233–240.

[44] Paolo Malinverno, Kimihiko Iijima, M. O. J. S. S. P. A. J. Gartner magic quadrant for full life cycle api management. *Forrester* (Sep 2020).

[45] Pappas, C., and Williams, I. Grey literature: its emerging importance. *Journal of Hospital Librarianship 11*, 3 (2011), 228–234.

[46] Pazos Corella, V., Chalmeta Rosalen, R., and Martinez Simarro, D. Scif-iris framework: a framework to facilitate interoperability in supply chains. *International Journal of Computer Integrated Manufacturing 26*, 1-2 (2013), 67–86.

[47] Randy Heffner, Christopher Mines, A. L. K. H. The forrester wave™: Api management solutions, q3 2020. *Forrester* (Aug 2020).

[48] Rao, N. R., and Sekharaiah, K. C. A methodological review based version control system with evolutionary research for software processes. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies* (2016), pp. 1–6.

[49] Ska, Y., and Syed, H. H. A study and analysis of continuous delivery, continuous integration in software development environment. *SSRN Electronic Journal 6* (09 2019), 96–107.

[50] StackOverflow. Developer survey of 2020, 2020.

[51] Ståhl, D., and Bosch, J. Experienced benefits of continuous integration in industry software product development: A case study. In *The 12th iasted international conference on software engineering,(innsbruck, austria, 2013)* (2013), pp. 736–743.

[52] Stahl, D., Martensson, T., and Bosch, J. Continuous practices and devops: beyond the buzz, what does it all mean? In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (2017), IEEE, pp. 440–448.

[53] Thompson, L. A. Grey literature in engineering. *Science & Technology Libraries 19*, 3-4 (2001), 57–73.

[54] Venable, J. A framework for design science research activities. In *Emerging Trends and Challenges in Information Technology Management: Proceedings of the 2006 Information Resource Management Association Conference* (2006), Idea Group Publishing, pp. 184–187.

[55] Venable, J. The role of theory and theorising in design science research. In *Proceedings of the 1st International Conference on Design Science in Information Systems and Technology (DESRIST 2006)* (2006), Citeseer, pp. 1–18.

[56] W3. Contrast (minimum): Understanding sc 1.4.3. W3 org.