Developing Application Programming Interface (API) Generator for Role-Based Access Control System in Social Networks

By

Raneen Waleed AL-karaki

Supervisor

Dr. Khalil Massri

Co-supervisor

Prof. Mario Macedo

2021

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree at Hebron university

Developing Application Programming Interface (API) Generator for Role-Based Access Control System in Social Networks

By

Raneen Waleed Al-karaki

This thesis was defended successfully on November 2021 and approved by:

Committee Members		Signature
1- Dr.Khalil Massri (Hebron university)	Supervisor	[illegible]
2- Dr.Mario Macedo (Atlantica university)	Co-Supervisor	[illegible]
3- Dr.Mohanad Jabari (Hebron university)	Internal Examiner	[illegible]
4- Dr.Mamoun Abu Helou (Al-Istiqlal university)	External Examiner	[illegible]

AKNOWLEDGEMENT

I would like to thank all the people who have helped me during my thesis research. Special thanks to my supervisor Dr. Khalil Massri for his immense support and guidance throughout my master’s degree.

Special thanks to Dr. Mahmoud Awad for his immense support and guidance throughout my master’s degree.

My sincere appreciation goes to my parents for their unconditional support.

Publication

This research is a supplement to a published scientific article by the teamwork titled "User as a Super Admin: Giving the End-Users Full Control to Manage Access to Their Data in Social Media Networks" (Awad et al., 2021). This was presented at the "International Symposium on Electrical, Electronics and Information Engineering (ISEEIE 2021)" in South Korea. And the article was classified as a featured article at kudos site which is affiliated with ACM**.**

TABLE OF CONTENTS

LIST OF TABELS

LIST OF FIGURES

List of ABBREVIATIONS

API	Application Programming Interface
SN	Social Networks
RBAC	Role-Based Access Control
REST	Representational State Transfer
CDMI	Cloud Data Management Interface
UI	User Interface
MVC	Model-View-controller
ORM	Object Relational Mapping
PD	Program Development

Developing Application Programming Interface (API) Generator for Role-Based Access Control System in Social Networks

By Raneen Waleed Al-karaki

Abstract

With the huge expansion of social networks, they become an important part of users' daily life everywhere, where users have large amount of data stored in these networks. these networks provide users with the tools and settings to control their privacy and security. Even though these settings give the users control over how to manage access to their data, they remain according to the specifications of the network providers rather than the user himself/herself. Role-based Access Control (RBAC) is a method used widely in managing user authorizations and restrictions in computer systems.

This thesis research presents a new system called Etqan. Etqan is an API generator, that can create a fully RESTful API for any system with a granular full access control sub-system. Etqan will allow the users to easily generate APIs to access their own data and they can build the access control within these APIs. This research presents the design and develop the Etqan system and also validates, assesses, and tests the performance of the system.

As a result, the Etqan system enables the users to act as super admins over their own data, where the actual network provider has no control over the users' data, this will improve user privacy and security as the users are the only person who can grant access to any other user to access the data.

الملخص

مع التوسع الهائل للشبكات الاجتماعية عبر الإنترنت، أصبحت جزءًا مهمًا من حياة المستخدمين اليومية في كل مكان، حيث يمتلك المستخدمون كمية كبيرة من البيانات المخزنة في هذه الشبكات. توفر هذه الشبكات للمستخدمين الأدوات والإعدادات للتحكم في خصوصيتهم وأمانهم. على الرغم من أن هذه الإعدادات تمنح المستخدمين التحكم في كيفية إدارة الوصول إلى بياناتهم، إلا أنها تظل وفقًا لمواصفات مزودي الشبكة بدلاً من المستخدم نفسه. التحكم في الوصول المستند إلى الدور (RBAC) هو طريقة تستخدم على نطاق واسع في إدارة أذونات المستخدم والقيود في أنظمة الكمبيوتر.

يقدم هذا البحث في الأطروحة نظامًا جديدًا يسمى إتقان .هو منشئAPI ، يمكنه إنشاء واجهة برمجة تطبيقات RESTful بالكامل لأي نظام للتحكم في الوصول الكامل. سيسمح إتقان للمستخدمين بإنشاء واجهات برمجة التطبيقات بسهولة للوصول إلى بياناتهم الخاصة ويمكنهم بناء التحكم في الوصول داخل واجهات برمجة التطبيقات هذه. يقدم هذا البحث تصميم وتطوير نظام إتقان وكذلك التحقق من أداء النظام وتقييمه واختباره.

ونتيجة لذلك، يمكّن نظام إتقان المستخدمين من العمل كمسؤولين فائقين على بياناتهم الخاصة، حيث لا يتحكم مزود الشبكة الفعلي في بيانات المستخدمين، وهذا من شأنه تحسين خصوصية المستخدم وأمانه لأن المستخدمين هم الشخص الوحيد القادر على منح حق الوصول إلى أي مستخدم آخر للوصول إلى البيانات.

Key words: Social Networks, Access Control, Role-Based Access Control, API, REST API

CHAPTER 1

1. INTRODUCTION

1.1. Research Background

During the last two decades, Social Networks (SN) such as Facebook, Twitter, Instagram are some of the most in-demand online services that are provided on the Internet(Boyd & Ellison, 2007). These SN attract users because they allow communication opportunities with new people around the world that they are known with or even unknown to them, where they can share information with their network of friends (Boyd & Ellison, 2007).

Users take the advantages of this service to share all kinds of digital content within their social network. These social network-sharing functionalities are extremely powerful and engaging for further social interaction. and with the expansion of social networks, there has been an increase in worry regarding the privacy of those who use them. While sharing information on the web is a voluntary activity on the part of users, consumers are frequently uninformed of who has access to their data and how it may be utilized. Data privacy is defined as "resistance against unlawful penetration."(Zheleva & Getoor, 2011.)

In general, data privacy refers to a person's ability to choose when, how, and to what extent personal data about them is shared with or conveyed to others. This personal information can include a person's name, address, phone number, and online or offline conduct. Many online users desire to regulate or avoid some sorts of personal data collecting, much as they might want to exclude persons from a private conversation (Barker et al., 2009).

The importance of data privacy has grown in tandem with the rise in Internet usage over the years. In order to deliver services, websites, software, and social networks platforms frequently need to collect and store personal data about users. However, some programs and platforms may exceed users' expectations in terms of data gathering and utilization, leaving users with less privacy than they anticipated. Other apps and platforms may not have proper controls in place to protect the data they gather, which can lead to a data breach that jeopardizes user privacy (Barker et al., 2009).

Social Networks are a cause of serious privacy and security problems because sharing control in social networks is not in the user's hands. This in fact becomes a serious threat to user privacy because the content shared in these platforms is controlled and owned by the social network itself, and can easily be exposed to a wider users in just a few seconds without user awareness (Beye et al., 2012) . In 2015, a report by the Protection and Privacy awareness organization on social networks (SN) showed that Facebook monitored the browsing histories of users, including users who no longer had an account and those who preferred not to be followed by the site. Such direct privacy violations will lead to less protected information for users on Facebook and users need to be in charge of who has access to their data(Vecchiato et al., 2010). Additionally, the government can spy on individuals online; By viewing their Facebook data, and some Internet users, due to their lack of knowledge might be unintentionally compromising their privacy. Quite often, they are not aware that their profiles are readily available to basically every other user. They often post details about their private lives, mistakenly believing that this information will remain well-protected within their friends' circle. Therefore, there are strong demands for a means that enable users to control their privacy in social networks.

Access control is the most important technique for information security prevention and protection to ensure that system resources aren't misused (Cai et al., 2019). So, resources can be shared more safely with the help of access control, which prevents unwanted users from gaining access to resources.

Role-Based Access Control (RBAC) model enables assigning resource permissions to a different type of users according to their roles in their organization. The model became of great popularity as an approach for managing access to data. The RBAC model usually forms the access control policy with the combination of users and permissions. The role is the basic element in the RBAC model and the whole access control policy is formulated to address this element. A role undertakes different presentations and can be mapped to different scenarios(Fuchs et al., 2011). RBAC limits access to the information by allowing users to access only the part they need to perform their tasks and prevent them to have access to information that they don’t need or belong to them. RBAC works on the principle of assignment of roles to the users instead of giving access to individuals. In a network, the roles refer to the level of access that users are given. Access is dependent upon many and numerous factors, such as job competency, responsibility, authority & securing confidential information. Roles consist of attributes; Attributes of the role are the properties of the job they are assigned and the type of work they are allowed to do and the kind of data they are allowed to access. It also consists of the type of data and what data that they are not allowed to access. The role is defined in such a way that, the semantic concept around which the policy, rules and regulations of access control are made. The role brings together the user and permissions.

So, in this research, we will adopt on role-based access control to allow the user to control his/her data in social networks.

There are many ways to give users control over access to their data, the first method is server-side by customizing site access control through the site or application. The second method is by client-side by adding access control as a plugin in the browser but they need a plugin for each browser and you need the approval to add it. Another method is by providing other parties API functionality by which access control could be determined. This last method, which is the API, is the best solution, as it works on all platforms, and this is the method that we will use in this research.

1.2. Problem Statement

In SN Users don’t own their data (Gurses et al., 2008), the SN providers control the data and provide users with facilities to manage access to their data, these facilities are different from one SN network to another and usually, they are not compatible with each other, users may not be fully aware of these features. If they change their SN provider, they cannot take their own data with them, and they may be introduced with a new user interface that they are not familiar with to manage access to the data.

The options that are available for users to manage access control to their own data are only managed by the SN provider, these options are limited with what the SN is offering, and these systems are usually not clear enough for ordinary users to understand them in detail.

When the users join social networks platforms, users agree to a set of general terms and conditions. A large part of the relationship is dependent on the company's: Privacy policies, Marketing materials, and Standards and practices(Lynn et al., 2021).

The high-profile data breach involving Facebook and Cambridge Analytica is an example of a company's policies and practices being violated. Many U.S. jurisdictions have some type of data breach notification regulation requiring corporations to divulge such leaks, thus companies like Facebook can face fines if they do not notify consumers of these breaches.

So according to U.S. jurisdictions privacy law users should own their data and be able to take their data with them from one SN to another, they should be the only authority that determines access control the way they desire(Lynn et al., 2021).

To achieve these goals, this research introduces a new open-source system called Etqan. Etqan is an API generator, that can create a fully RESTful API for any system with a granular full access control sub-system. Etqan will present a solution to the mentioned problems as it allows users to easily generate any API and control the access of his/her data. Users can generate the access control easily within these APIs. This research aims to design and develop the Etqan system and to validate, assess and test the performance of the system.

1.3. Research Questions

1. How to design and develop a system that can generate a complete set of APIs that can enable ordinary users for using it efficiently to share their data and control access.
2. How the generated API system can be customized and deployed to enable normal users to control access to their data, how the problem of making this API generate accurate business logic and maintain the privacy and security of the users at the same time.
3. Such a system will impose a challenge of how to make novice users use this system to control access at a high level.

1.4. Importance of the Research

As with the expansion of social networks, and the amount of time users spends on them increased, users are now sharing more data about their life on these platforms, users are not aware of privacy issues, security measures or possible threats to their data on these SN (Ali et al., 2018). This research aims to improve user privacy and security when sharing and exposing their data on SN. When users share their data on any current SN (Facebook for example), these SN acts as the owner of the user's data and have full control over it, the users cannot share or re-use the data without the permissions from these SN, and if any SN doesn’t provide the users with a public API facility it will not be possible for the user to carry their data to other platforms. This research focuses on how to make users the only owner of their data and the only administration authority that can control who to access the data and how. On the other hand, the research will provide new potentials when they no longer need to implement third party APIs and authenticate their applications to access SN, but the users will come to their apps, then users will authenticate these apps and decide how they would like to share their data and assign roles that the app provides, therefore application developers will be only concerns on how to design their application and how can users interact with it, they should also provide a set of roles that can perform actions in the app, these roles will be re-used by users to give to their people and give them the required access control.

To better highlight the importance of our system, we distribute a questionnaire among most of our relationships on social networks. According to the survey results, most social media users are concerned about the security and safety of their data on these sites, and they are unaware that this data is the property of the site owners, who can dispose of it without their permission. In section 3.4, we will go over the questionnaire.

CHAPTER 2

2. TEORITICAL FRAMEWORK AND LITERATURE REVIEW

2.1. INTRODUCTION

The main purpose of this chapter is to introduce relevant background material. In Section 2.1 we give a brief introduction to some basic concepts and earlier developments in access control, and API. In particular, we introduce the types of access control – Role-based access control, and RESTfull API. In Section 2.2 we introduce a review of role-based access control and RESTfull API review. which is fundamental to the remainder of this thesis. The chapter comes to a close with a summary of the literature review.

2.2. TEORITICAL FRAMEWORK

2.2.1. Access Control

Access control is the most important technique for information security prevention and protection to ensure that system resources aren't misused (Cai et al., 2019). So, resources can be shared more safely with the help of access control, which prevents unwanted users from gaining access to resources. While developing access control technology, various typical approaches have been presented, with the most representative models being Discretionary Access Control, Mandatory Access Control, and Role-Base Access Control, due to the necessity for data security in the system.

- Discretionary Access Control (DAC): When an object's owner uses discretionary access control (DAC), he or she decides whether other subjects can access the object (Kühnhauser & Pölck, 2011). Ownership of controlled objects is central to DAC's design, with access privileges being managed by their owner. A more flexible permission policy can be configured with autonomous access control. The object owner can easily and directly grant or deny permissions.

- Mandatory Access Control (MAC): According to the system, subjects and objects are assigned to categories and levels based on a matching relationship (Y. Wang, 2021). MAC has two different implementation models. According to BLP, an object of low security must be able to be written to, while an object of high security can only be read by the subject of low security. In the Biba paradigm, a high-security-level subject must write permissions to an object with a low-security level, while a low-security-level subject can only read permissions to an object with a high-security level.
- Role-Based Access Control (RBAC): it is a way to restrict access for users by assigning them a role (Osborn et al., 2000). RBAC assigns a role to each user and then allows them to access all of that user's access permissions. Using the RBAC model, subjects and objects may be separated. Roles fulfill the concept of authorization groups. Permissions can be granted or revoked without affecting the subject or object.

In this research, using role-based access control, we will provide a way to protect user data in social networks (SN).

2.2.2. Social Networks (SN)

social networking has come to mean individuals using the Internet and Web applications to communicate in previously impossible ways. This is partly due to a cultural shift in the usage and capabilities of the Internet itself.(Wazza et al., 2008.)

Social networks offer unlimited opportunity to engage with those who share your cultural, political, religious, and other interests. Extending your social network beyond your familiar circle of friends can have unexpected benefits, as social networking activities transform into socioeconomic opportunities, bringing new ideas through shared information and unexpected opportunities in the form of a job, an apartment, or even a partner. The Internet gives tools for establishing, controlling, and capitalizing on those networks, allowing you to form an initial relationship with someone you've never met in person, who not only improves but may even change the course of your life.(Wazza et al.,2008.)

2.2.3. RESTFUL API

An Application Programming Interface (API) is an architectural design approach for an application to communicate with other applications. While A RESTful API is based on representational state transfer (REST), which is an architectural approach to communicate over using web services. REST uses HTTP requests to access and use data. RESTful API has four main methods: GET, PUT, POST, and DELETE, which are used to read, update, create and delete resources. A RESTful API handles clients requests, it breaks down a request into a series of smaller modules. Each module handles a part of the request. This modularity provides developers with a lot of flexibility, but it can be challenging for developers to design their REST API from scratch. Currently, several organizations provide models for developers to use; the models provided by Amazon S3, Cloud Data Management Interface (CDMI), and OpenStack Swift are the most popular. The state of a resource at any given timestamp is called a resource representation. A RESTful API uses existing HTTP methods defined by the RFC 2616 protocol, such as:

- GET to retrieve a resource;
- PUT to change the state of or update a resource, which can be an object, file or block;

- POST to create that resource; and
- DELETE to remove it.

In a RESTful API system, users request resources through a network in a black-box method where the implementation details are hidden. It is important to understand that all calls are stateless; where nothing is retained by the RESTful service between executions. Because the calls are stateless, REST is useful in cloud applications. Stateless components can be easily redeployed if they fail, because the new attempt will not depend on the previous one as there nothing needs to be stored, and any request can be directed to any instance of a component. The RESTful model is good for cloud computing because accessing a service through an API call depends on how the URL is constructed.

A true RESTful API follows the REST architectural constraints which are (Fielding et al., 2017):

- Resources should be uniquely identifiable through a single URL, and only by using the one methods of the http protocol, such as DELETE, PUT and GET.
- There should be a clear distinguish between the client and server. The users interface and request-gathering is only the concern of the client. Data access, workload management and security are the server duties. This loose coupling of the client and server makes each to be developed and maintained independently.
- All client-server operations should be stateless, and any state management that is required should only dealt with on the client, not the server.
- All resources should allow caching unless explicitly indicated that caching is not possible.
- REST allows for an architecture that is consist of multiple layers of servers.

- The server will often send back a response to represent the resource in a form of structured data such as XML or JSON format. However, in some occasion, servers can send executable code to the client.

There are many challenging in developing a RESTful API system, these challenges include:(S. Wang et al., 2014)

- Endpoint consistency where URI of endpoints should be consistent by following web standards, which could be difficult to manage.
- API versioning where endpoint URLs shouldn't be invalidated when used internally or with other applications.
- The number of returned resources can increase in size in time, adding to increased load and response times.
- because REST uses URL paths for input parameters, determining URL spaces can be challenging.

There are also some security challenges for an API system that include:

- blocking access from unknown IP addresses and domains
- validating URLs
- blocking unexpectedly large payloads before parsing them
- logging requests
- investigating failures.
- Use of different authentication methods such as HTTP basic authentication (a base64-encoded representation of username: password string), API keys, JSON Web Tokens, OAuth 2.0.
- requests may have more data and metadata than needed or more requests may be needed to obtain all the data.

Testing API can be a long process to set up and run. Testing can also be done in the command line with the utility Curl. Parts of the testing process that may be challenging include:

- Initial setup
- Schema updates
- Test parameter combinations
- Sequence API calls
- Validate test parameters
- System integration
- Define error codes and messages.
- With error codes, it is more of a common practice to use standard HTTP error codes. These are recognized by clients and developers more often.
- Error handling may not have a way to distinguish if a response is successful or not besides parsing the body or checking for an error.

2.2.4. Haxe language

Haxe is an open-source cross-platform programming toolkit that allows programs to be compiled for many target languages or systems. It is made up of three parts: the Haxe language, the Haxe compiler, and the Haxe standard library(Galić, 2015.).

The Haxe programming language is a high-level programming language that supports both functional and object-oriented programming ideas (e.g., type inference, nested functions, and recursion) (e.g., classes, interfaces, enumerators, getters, and setters). Haxe is statically typed, but by using a Dynamic data type to represent untyped data at design time, it retains the flexibility of dynamically typed languages. The Haxe language combines elements from the Haxe compiler's supported languages.

Conditional compilation abstracts away language-specific incompatibilities, allowing for the creation of specific code based on compilation parameters. As a result, conditional compilation is critical in cross-platform programming. The Haxe language syntax is based on the ECMAScript standard, with some deviations where needed. Haxe is a compiled language, unlike other ECMAScript languages like Javascript(Galić, 2015.).

The Haxe cross-compiler is a command-line utility that converts Haxe source code into target language or platform source code or bytecode. Supported languages and platforms as of Haxe 3.1.3 include Flash, Neko, JavaScript, ActionScript 3, PHP, C++, Java, and C#, with Python support guaranteed for version 3.2. A Haxe program's base class is lexically scanned by the compiler, beginning with the entry point represented by a static main function. During the scanning process, the compiler looks for new class name occurrences, which are also lexically scanned. The code is then parsed and type-checked before being subjected to macros, optimizations, and transformations, yielding a typed abstract syntax tree (AST). Depending on the language or platform targeted by the compiler, the abstract syntax tree is subsequently converted to either source code or bytecode(Galić, 2015.).

Haxe's standard library includes a general-purpose API, a system API, and target-specific APIs. The general-purpose API includes classes that describe data types, data structures, and algorithms that can be used on any target. Because the system API incorporates file system and database APIs, it can only be accessed when compiling to targets that enable such operations.

C++, C#, Java, Neko, and PHP are among the targets. Target-specific APIs contain operations that are unique to the targeted language or platform and can only be accessed when compiling to that target.(Galić, 2015.)

2.3.LITERATURE REVIEW

An access control scheme for Social Networks (SN) was suggested by (Pang & Zhang, 2014), they focused on public information, and took Facebook as an example. They aimed to show that through user's attributes, common interests, and activities within public information they can group users. They defined public information as the information that exists in public domains such as Wikipedia and Bing Map. Pang and Zhang model contains information of:

- Users and their social relationships.
- Public information and their connections.
- Links between users and public information.

They created a graph for users and their relationships and another graph that depicts the links among public information. However, the links between users and public information are established with user interaction with public information, after establishing the links they used a hybrid logic to define the access control scheme. The question about their policy is whether the system would generate an access control policy out of the user's intention? And from a usability perspective, how it will be easy for a user to establish access control based on their formulas.

A novel user-to-user relationship-based access control (UURAC) model for SN systems that utilizes regular expression notation for policy specification was proposed by(Cheng et al., 2016) , they presents two-path checking algorithms to determine whether the required relationship between users for a given access request exists. After that, they validated the feasibility of such an approach by implementing a prototype system and evaluated the performance of these algorithms. Their model components consist of:

- Accessing user (UA): a human user who performs activities.
- Actions: an abstract function initiated by accessing the user against the target.
- Target is the recipient of an action, can be either.
- Target User (UT).
- Target Resource (RT), which has a relationship with a controlling user (UC).
- Access Request denotes an accessing user's request of a certain type of action against a target.
- Policy defines the rules according to which authorization is regulated.
- User to user relationship (U2U).
- User to resource relationship (U2R).

Their solution provided a solid foundation for a more advanced relationship based on access control in the future. In (Sachan & Emmanuel, 2011) a bit-vector transform-based access control mechanism for multimedia contents in social media was applied. The study showed the efficiency of the proposed approach by means of Mathematical analysis and experimental results. Access rights associated with the contents were organized into an M-dimensional space. Each dimension corresponds to a specific access policy.

An approach to facilitate the defense of shared data associated with many users in SNs was proposed by (Praveena et al., 2014). Their proposal contained an access control replica to capture the essence of multi-party agreement requirements, along with a multiparty strategy requirement scheme and a policy enforcement mechanism They introduced a proof-of-concept prototype of their model as part of an application in Facebook by creating an experimental social network using synthetic data which used to test the efficacy of the semantic reasoning-based approaches. They demonstrated that a group of users could collide with one another to manipulate the final access control decision.

A system that relies on an automatic semantic annotation mechanism was proposed by (Imran-Daud et al., 2016). The system used knowledge-based and linguistic tools to associate a meaning to the information before it is published, their mechanism automatically detects the information that are sensitive according to the privacy requirements of the publisher of data, with regard to the type of reader that may access such data.

An access control model which supports relation names and properties that is being defined and customized by users was introduced by (Tapiador et al., 2017.). These customizations they included are unidirectional and flexible in such a way that doesn't constrain access control design. The key work they introduced is the application of their work to social media. Actors define their custom relations with other users in social media such as a friend or a partner, these defined relationships are equivalent to roles, after that. Actors can assign permissions to relations, which can include read wall or post to wall. Additionally, actors can make ties using those relations. They were able to test their system in a videoconference application, where they defined relationships between social entities including individuals and social events.

(Carminati et al., 2010) argued that the access control mechanism of social media networks needs to be improved, they created an experimental social network to address the limitation in an access control policy, they equipped their social network with a fine-grained social network access control model, and then they presented experimental results for the length of time access control can be evaluated using this scheme. They have demonstrated that existing social networks require some sort of data partitioning for semantic inference of their access control to be reasonable in its speed and memory requirements.

(Beato, Kohlweiss and Wouters, 2011) on the other hand, they argued that the control of accessing the data should not be in the social media network itself, but with the actual owners of the data, and therefore accessing to the data should be platform-independent. They proposed a system that defines access control restriction to the data which uses encryption to enforce access for users' private information based on their privacy preferences. They have implemented this system as a Firefox extension, this extension knows about the users' access control preferences. This way users are able to execute control over their data with no third-party influence, by allowing each user to have its own trusted circle, so any users within that circle can have access to target content.

Users on social media networks have their data exposed to other users without their consent or agreement, for example, the user of a social network would be exposed to the network providers themselves, other users in the same network, users from outside the network. Normal users will not be completely knowledgeable about the flaws that exist in their access control. To assist users in learning about access to their data and managing their access control.

Many researchers have used the API, developing RESTful API systems to build APIs from web-based HTML documents, while other researchers are developing systems to construct RESTful APIs to avoid limitations on the leading developer to make creating and maintaining REST websites easier for developers.

(Ed-douibi et al., 2017) proposed A methodology that enhances the Model-Driven Engineering (MDE), this methodology can be used to create RESTful services automatically. They called their suggested system EMF-REST which takes data models as input and then the system which uses famous libraries and outputs the corresponding web APIs based on the standard RET principle, thus Enhancing its understanding and maintenance. EMF-REST also includes model and web-specific features to provide the created API with model validation and protection features respectively. For Software developers, this approach makes the web development process more agile by offering ready-to-run Web APIs from data models. Furthermore, the methodology provides MDE professionals with the framework for designing cloud-based modeling solutions and improved cooperation.

AutoREST is an automated approach presented by (Cao et al., 2017), the goal of this approach is to generate a REST API Specification from a REST API HTML Documentation automatically, this shows that AutoREST inputs the REST API Root Page Documentation of a specified Web service and removes the four compulsory components that make up the OpenAPI specification: the base URL, the path templates, the HTTP verbs, and the formal parameters associated with it. And then returns the OpenAPI Specification created, AutoREST executes three phases, the first phase is to define all pages in the HTML documents. This phase is meant to identify all web pages that could be defined by the REST API, and the second phase is to Identify pages of useful or useless documentation. The purpose of this phase is to choose only web pages that contain valuable information to create a specification for the REST API, and the last phase is a data extraction and REST API Specification generation, The validation of this method indicates that AutoREST has very good results. It is less effective for randomly chosen web services, primarily because the supplied HTML documentation is not standardized like one of the most common web services.

Kaang is a RESTful API generator described in (Queirós et al., 2020) . Kaang's main aim is to reduce its effect by automating the workflow of the RESTFul service (e.g., files structuring, boilerplate code generation, dependencies management, and task building), based on the input of the user's and their feedback and on a sequence of templates, Kaang constructs the key API material that supports developers management, test paths, resource description, data models etc. To provide the end-users of the generator an increased degree of trust.

Two groups of users benefit from this type of generator: lets beginner developers minimize their learning curve when confronting new frameworks and libraries typically seen on the digital web and accelerates the works of developers who escape both tedious and bureaucratic jobs. Kaang further supports the good values of production through the addition of automated testing and reporting.

To generate RESTful API using kaang, the developer must only open the command line and invoke the generator in order to access to it. The API is customized based on the input data of the developer during the application generation process, the use of a reacting Interface or a your favorite API tester, a checked and documents RESTful API is created and consumed (e.g., Postman). The programmer will obviously have to change it, i.e. add routes or new functionality. However, it's necessary to create the whole workflow for software development and minimize all the hard first activities, freeing the developer from more important coding aspects of the program.

(Sohan et al., 2015) implemented a new methodology for automating RESTful API documentation using the HTTP proxy server, which would otherwise entails a considerable amount of manual process. The solution based on proxies supports advanced functionality such as automatic RESTful API documentation with executable samples of API calls, various version support, SaAS customization and collaboration, and self-hosted platforms. Current literature on API documentation proposed these features. We believe that a proxy server approach offers a logical alternative to the documentation dilemma of the RESTful API.

Similar to our system where we use API to access user data in social networks, the Solid project introduced by (Mansour et al., 2016) introduces a decentralized platform for social Web applications. users' data is managed independently of the applications, where the data ownership remains for the user who created it, the user stores their data in a Web-accessible personal online datastore or as they call it a pod. Each user can have one or more pods from different providers. Applications access data in users' pods using well defined protocols, and a decentralized authentication method, the users also provided with a complete access control mechanism that guarantee the privacy of the data. In this decentralized architecture, applications can operate on users' data wherever it is stored.

You can control who and what apps may access your data using Solid's Authentication and Authorization mechanisms. You have the ability to grant or withdraw access to any portion of your data as needed. As a result, you can do more with your data since the applications you choose can have access to a broader and more diversified collection of information.

And, just as you may share your data with others, they can do the same for you. This results in rich and collaborative experiences across a mix of personal and shared data.

Solid apps use the Solid Protocol to store and access data in Pods.

Different apps within the interoperable Solid ecosystem can access the same data rather than having separate data silos for the applications. Instead of entering your email address into your bank's statement notification service, your phone's billing service, and so on, you may save this information in your Pod and allow these disparate services/applications permissions to view your email information.

Php access control list (ACL) system it is an Access Limit Lists enable an application to control access to its areas by providing a configurable interface for generating Permissions, Roles, and Resources, as well as assigning the established permissions to roles. This component is an implementation of an ACL, it makes it easy for you to get up and running with user authorization(Vasenin et al., 2020).

Php ACL system has several capabilities, including the ability to create Resources, Roles, and Permissions, as well as the ability to establish Permissions on Resources and give these Permissions to Roles. Role Inheritance Support, fully serializable, interoperable with any data source Simple to use and compatible with PHP v7.0+

2.4.DISCUSSION

From the above literature review we could summarize the following observation. Many academics have stated in earlier studies (Pang & Zhang, 2014) (Imran-Daud et al., 2016) that the access control mechanism of social networks could be improved, and they constructed an experimental social network to address the shortcomings in access control policy. Others claimed that control of data access should not be with the social network itself, but with the real owners of the data, and that data access should therefore be platform agnostic. Many academics (Cao et al., 2017) (Ed-douibi et al., 2017) have created RESTful API systems to create APIs from HTML texts on the web, and validation of this approach shows that AutoREST produces excellent results. However, it is less successful for randomly chosen online services, owing to the fact that the HTML documentation offered is not as standardized as that of a more popular web service.

A decentralized platform for social web apps has been developed by several academics. User data is maintained independently of apps (Mansour et al., 2016), and data ownership stays with the user who developed it, unless the user maintains his data in a personal online data storage accessible over the Internet, referred to as a pod. It is difficult to utilize such a system since it is dependent on a unique protocol.

In our research, we will create APIs using JSON files, By doing so we get the following advantages:

APIs will work across all platforms and devices. The API will be programmed to regulate access to data on social networks and will utilize the HTML protocol to access this IP, making it simple to use with any social networking site.

CHAPTER 3

3. SYSTEM DESIGN

The goal of this chapter is to go through the methodology that was utilized to set up the system, as stated in the section 3.1. In section 3.2, there is information about the overall design of the system and its components. The JSON file and its structure are covered in more detail later in section 3.3. Section 3.4 discusses the API structure, The Grant library, which will be established, is addressed in section 3.5. The outputs of the system are addressed further in section 3.6, the system is described in-depth, and ultimately the UML class diagrams are explained in section 3.7 and section 3.8.

3.1.Methodology

The project methodology was designed in many stages, these stages show in figure 3.1:

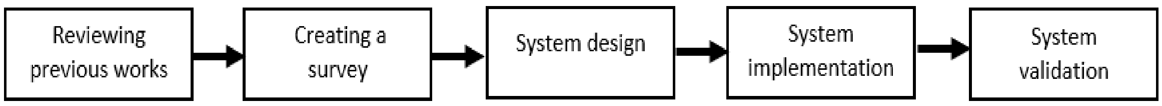


Figure 3.1: project stages

- The first stage is reviewing previous works, comparison, and determination of system requirements.
- The second stage is creating a survey and sending it to social networking site members to highlight the relevance of the research.
- The third stage is the system design, which includes the design of the system's inputs, structure, and outputs.
- The fourth stage is the implementation of the system.
- The fifth stage is testing the system and system validation.

3.2.Survey

To better highlight the importance of our system, we distribute a questionnaire among most of our relations on social networks. We selected questions to figure out the main goals of using the social networks by people. The types of information are usually posted on social networks. The degree of awareness about the security and privacy mechanism usually the users of online social networks have. In addition, questions to see whether users trust the owners of the social networks to store their data.

As shown in Figure3.2. The majority of users are using social networks to follow their friend's activities and share their photos with others (79.5%). However, in Figure3.3, 48.7% of them do not know precisely the exact users their data might be accessed by. In this same context, 43.6% of them said that their information can be accessed by people they do not know as shown in Figure 3.4.

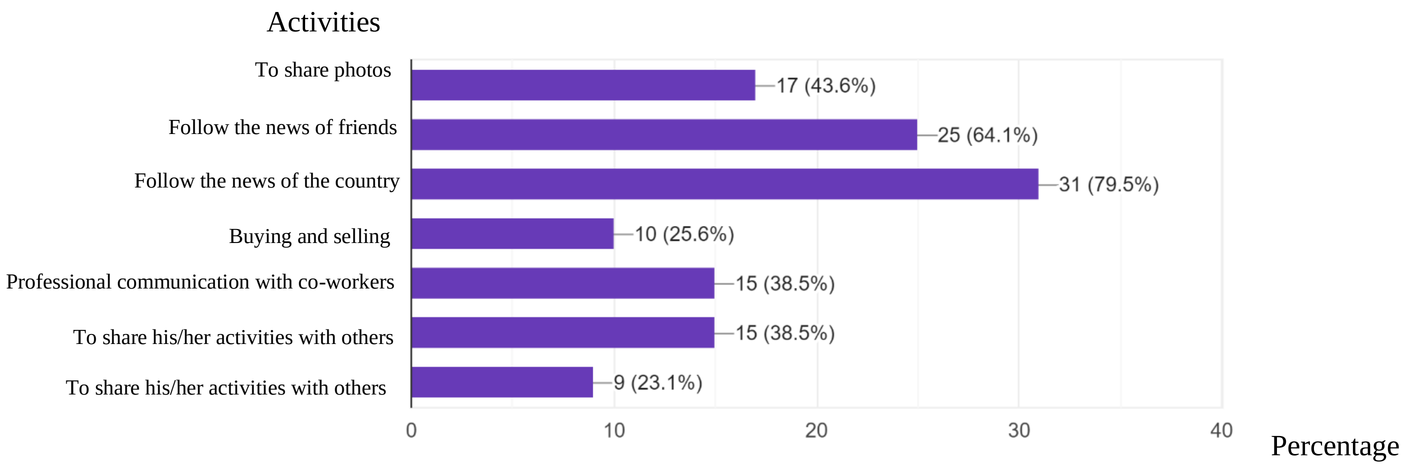


Figure 3.2: Percentage of participant activities

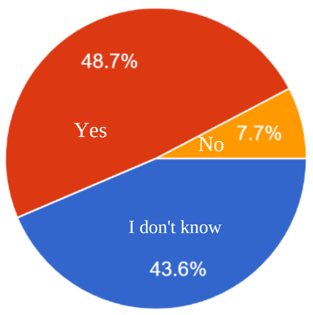


Figure 3.3: Percentage of user knowledge about the people who can see his information on social media.

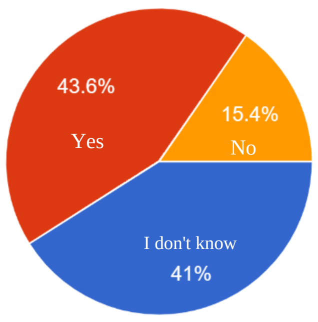


Figure 3.4: Percentage of user knowledge about the people who can see his information on social media and he don't know them.

Relating to how much people trust social networks, 66.7% of them said that they do not feel safe with their data when using social networks as shown in Figure 3.6. At the same time, 74.4% of them do not believe that social networks preserve their privacy on their data since as they believe that they lose ownership of their information at the moment they post it on social networks as shown in Figure 1.5.

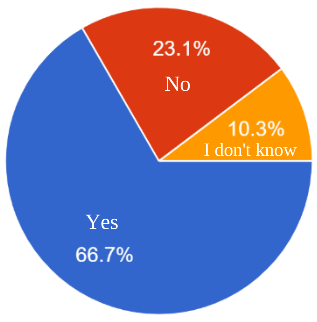


Figure 3.6: Percentage of user trust in social networking sites

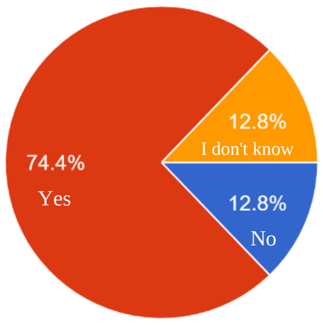


Figure 3.5: Percentage of user they don't think that their privacy is maintained by social networks.

For more details about the results of the questionnaire, you can see the Table in Appendix A.

3.3. The overall system

The main goal of system is to generate a RESTFUL application programming interface (API) from a JSON schema to host the generated API in any domain to use it for social networks, figure 3.7 shows the overall system components:

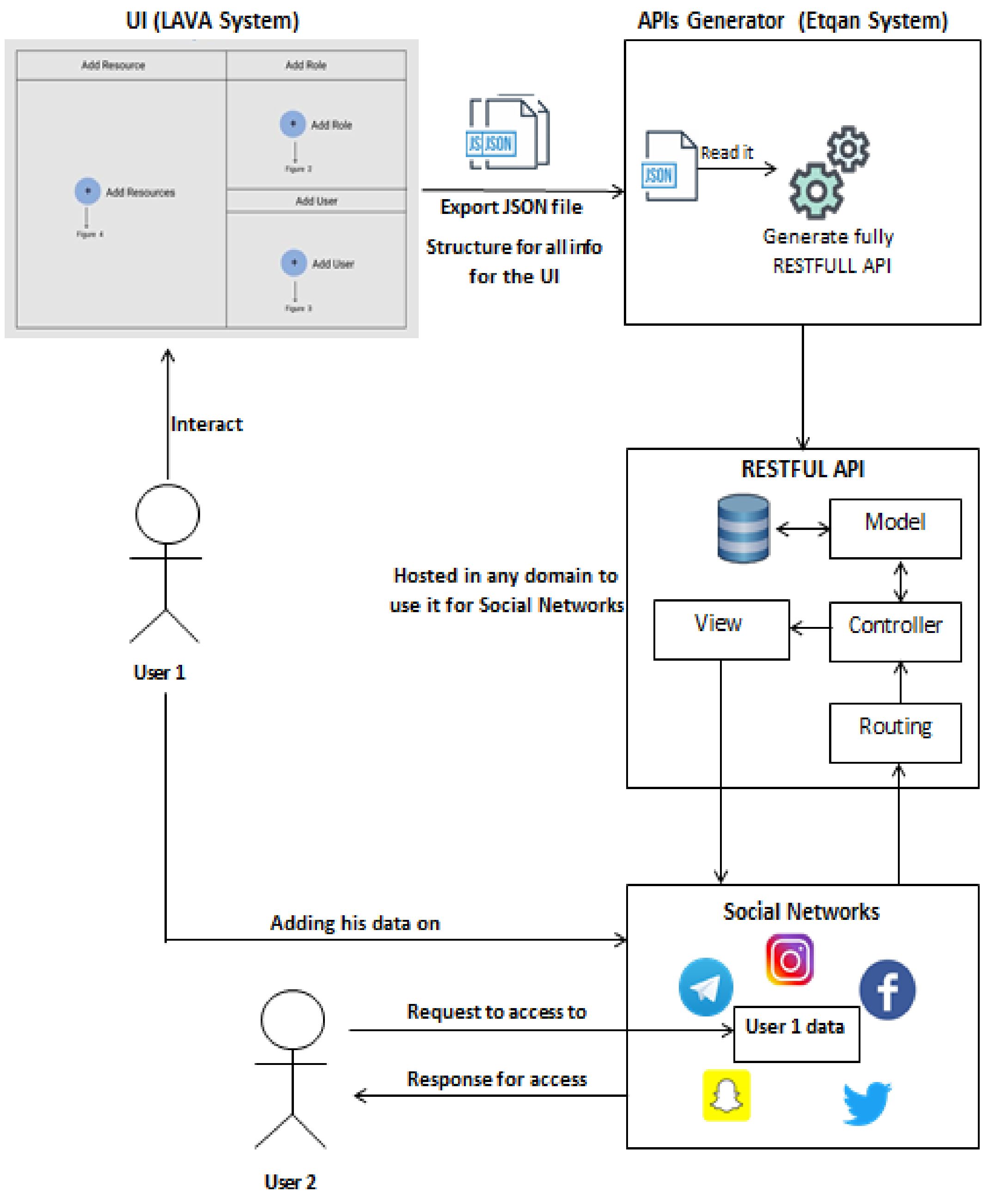


Figure 3.7: The overall system

The first component of Etqan is the user interface (UI) system which is called LAVA, Lava is a web system developed using React technology (Malley et al., 2009) and its main purpose is to provide the user with a user-friendly interface and help him/her creating the JSON schema. The JSON schema by itself is a high-level technical specification and cannot be provided directly by a regular user, therefore the Lava system facilitates the process for the user, and they can interact with a web-based GUI system to create the schema. After the user finished creating the visualization of the API system, they can set global parameters which will help Etqan to understand the user requirements.

Lava will convert a simplified entity-relationship (ER) diagram to the required JSON schema, this will be sent to Etqan as the main payload of a POST request.

LAVA is currently under development by a researcher in our team and till that we will assume that the JSON file is generated and ready to be used (we will manually generate it).

Figure 3.7 shows how Etqan subsystems work when receiving a request with a valid schema to generate the final API, the sub-system reads parameters from the JSON data to create a fully RESTful API system, which consists of different sub-components: Routing, Controller, Model, and views. This generated API system can be integrated by social networks, such as Facebook, any request to the data shared by the user will be done only through this generated API system, which has a complete system to generate any response to any request based on the given permissions.

For example, in Figure 3.1 there are two users, User 1 is the user who intends to share data on the social network, while User 2 is the user who is trying to access User 1 data, as shown. User 1 needs to interact with LAVA system to create an ER diagram that will be used to generate the final API that will guard access to whatever data he/she would like to share. Putting this into a real life context, assume User 1 would like to share family photos to Facebook, to protect his/her privacy and have full control over the data, User 1 uses Lava UI to create a system that visualize what data he/she would like to share and how it should be accessed, after that he/she obtains a fully RESTful API system that will act the main application to share these sets of photos online, now User 1 can add photos to the database through Etqan system using a POST request that he/she has given permission to himself/herself to create new records, the photos which he added using Etqan to the database now is managed by Etqan, User 1 doesn't need to know anything about the resources that he created, as he/she will have also an API to retrieve these resources and share them through Etqan to Facebook, the user will share Etqan API endpoints to these resources, if User 2 now tries to access these resources, he/she now be interacting with the social network itself, in this case Facebook, now through Facebook he will be sending a request to retrieve these resources, his request will include his role, for example User 2 is a friend of User 1, now Etqan will receive the request from Facebook Application on behalf of the User 2, Etqan will check what permissions were given by User 1 to the role "Friend on Facebook" and return the results based on that.

3.4.JSON file

We can divide the JSON schema into three main parts, the first one contains the general settings that are required for the API which includes the package name, project name, API version, the second part contains definitions for all resources and their attributes with validation rules, and, the third part includes the Access control schema which defines roles and assigns them permissions. The following figure3.8 show the main component of the JSON schema.

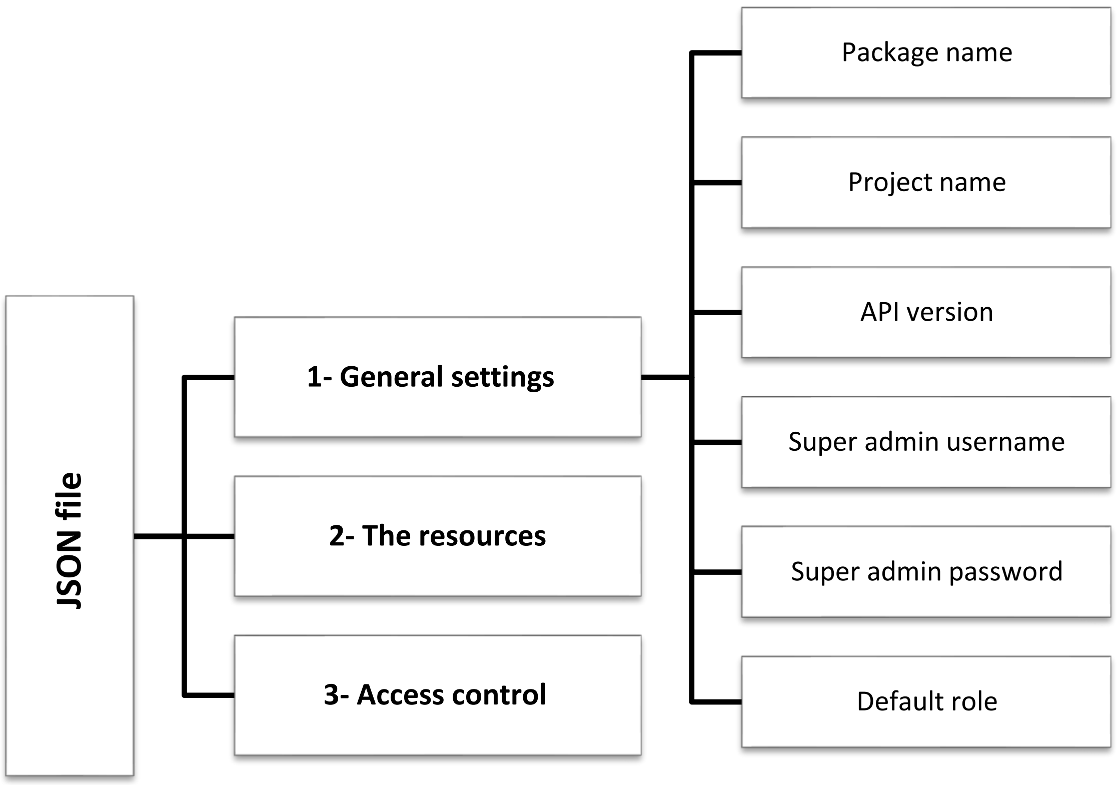


Figure 3.8: The components of the JSON schema

3.5. The structure for generated RESTful API

3.5.1. Restful API vs GraphQL

Restful API is usually used in a particular system to allow other systems to query some services from this system. other techniques recently are used for the same purpose such as GraphQL is the query language used to construct web service architectures. The language was created internally at Facebook as a solution to a number of issues they were having with traditional architectural styles like REST. Facebook has developed GraphQL to be an alternative to the well-known REST architecture, and it is distinguished by allowing the client program (Client Application) to specify the information it wants from the server, the latter sending only that information, and nothing else. To grasp the distinctions between GraphQL and REST, it's important to remember that endpoints are REST's primary abstraction[ref]. The following is an example of a REST request:

RESTful request

```
GET https://api.domain.com/products/22.
```

The response for this REST request is:

Response in JSON

```
{
   "id": "22",
   "name": "iPhone X",
   "price": "999",
   "color": "Gold"
}
```

REST endpoints can be thought of as low-level abstractions because they rely on HTTP resources to support queries (URLs, GET/PUT arguments, and so on). GraphQL, on the other hand, is a comprehensive data query language for building web-based services that are focused on high-level abstractions like schemas, types, queries, and mutations. The following is how the preceding REST query is implemented in GraphQL, we can query the product name and manufacturer name with a single HTTP request to the server:

GraphQL query

```
product(id: "22")
{
   name
   manufacturer {
        name
    }
}
```

The answer is supposed to be like this:

a GraphQL query result

```
{
    "data": {
        "product": {
            "name": "iPhone X",
            "manufacturer": {
            "name": "Apple"
            }
        }
    }
}
```

In comparison to REST, GraphQL needs less effort to build API inquire.

When it comes to the popularity of GraphQL versus RESTful API, the latter is undoubtedly the winner. The former is a new emergent technology on the block, attempting to keep up with an established API design style.

According to the recently released State of API 2020 Report, REST-based Open API is used by 82 percent of surveyed API practitioners and consumers, whereas GraphQL is used by only 18 percent. Although GraphQL has recently grown in popularity, it is still a young technology that may take some time to catch up to REST.

Compering GraphQL with REST API is compared in terms of predictability and versioning, the simplicity of GraphQL considerably increases its usability. When compared to REST, GraphQL requires less effort to incorporate remote service requests, according to a recent study. The tremendous flexibility of REST, on the other hand, is a neutral trait that might improve usability. Though GraphQL reduces the number of queries required by a typical application, while REST outperforms it in terms of caching.

To overcome the limitations, in this research, we will use a tried-and-proven technique that comes with robust native caching and authentication capabilities, which is REST could be the best option.

3.5.2. generated RESTful API

The API generator sub-system starts by reading the global settings from the JSON data, if the user doesn't provide a value, the system will generate a default one as shown in figure 3.9.

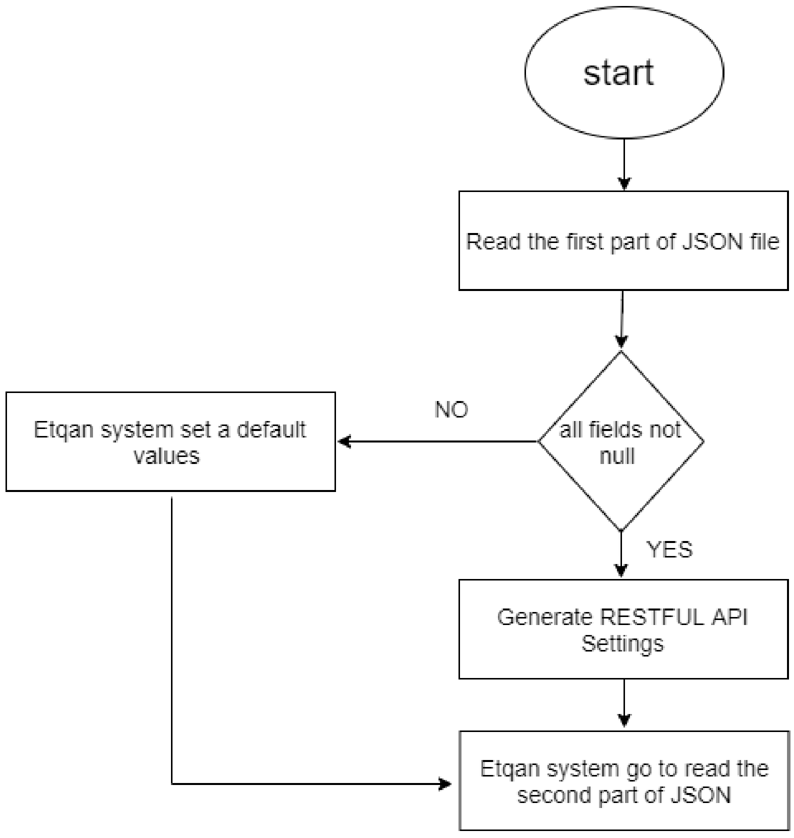

start
Read the first part of JSON file
all fields not null
NO
Etqan system set a default values
YES
Generate RESTFUL API Settings
Etqan system go to read the second part of JSON

Figure 3.9: Extract global settings

The code structure for the generated RESTful API is based on the Model-View-Controller (MVC) design pattern, which is a common pattern used to isolate presentation from data and logic, in MVC we have a model which is a class that contains the data about an object, View which is a class that contains all data required to view the object such as graphics, animation, sound, and video, and finally we have the controller which is a class that contains the logic of how this object interacts with the system (Osmani, 2012). All database classes are considered as models.

The models which represents the database tables and resources will be the next stage to be generated, each resource inside the JSON data Etqan will create a model for it with all the specified attributes, the attributes are the database fields, which include their names, data types, and constraints. Each model will have a physical database table it is corresponding to. The model will be the class that manages the communication between the Etqan system and the database, it will be responsible for the actual read and write operation to the database. However, there is another class that should be responsible for creating connections when necessary and closing them.

Etqan system also should have built-in definitions for frequently used data sets that can be reused across applications, such as generic data for gender, language, operating system, payment methods. And more, these data sets are rarely different in most applications and will be helpful to have them ready when they are needed.

After creating tables, Etqan also is responsible to manage relationships between these tables, these relationships either one-to-one, one-to-many, or many-to-many are determined and controlled by Etqan, the user can specify what data from each resource another resource will need and the Etqan system should be able to decide on the relationship type and its constraints.

Choosing Haxe as the programming language to create Etqan has many benefits including that Haxe provides a robust and unique Object Relational Model (ORM) that can create databases, tables, and relations from the code without the need to write any SQL statement, this will help in fact reducing the dependencies as all SQL logic will be abstracted out inside the application.

The following workflow diagram summarizes the process of creating the models and database tables inside the Etqan system:

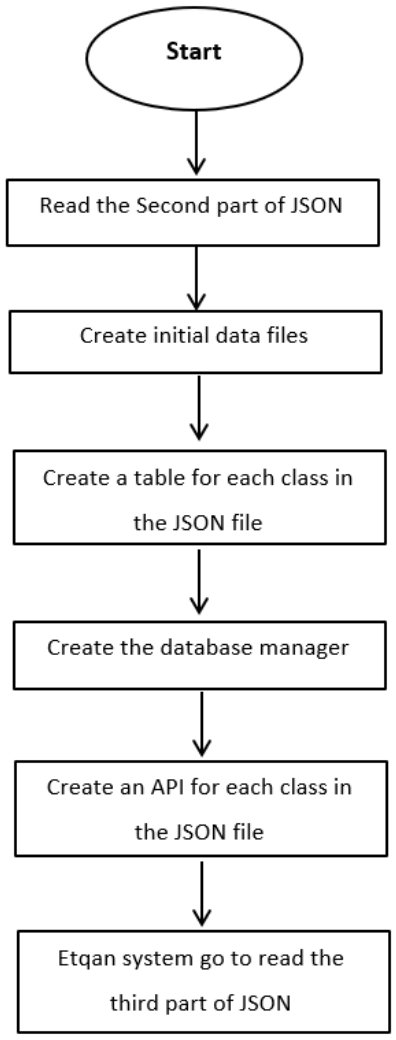

Start
Read the Second part of JSON
Create initial data files
Create a table for each class in the JSON file
Create the database manager
Create an API for each class in the JSON file
Etqan system go to read the third part of JSON

Figure 3.10: Creating the models and database tables

After creating model class, the Etqan system will create controller classes, for each model there will be a controller. The controller is the component that is responsible to communicate between the model and the API, the controller is responsible to receive data and validates it, authenticate the user, then pass the authenticated requests with validated parameters to the model, it also receives results from the models, filter the results based on the user role and send the data to the API class.

After the controller classes are ready, Etqan will generate the API classes, there should be an API class for each model, their main responsibilities are to communicate directly with the HTTP requests received by the clients, they perform routing logic, defining the method types (GET, POST, PUT, PATCH or DELETE) and where the request should be going, They also extract authentication data from the request header, extract parameters from header and body, and passes these data to the controller, the API also works as the first defense for any malicious acts, so they prevent SQL injections, they reject any bad requests or invalid type formats, they also receive the response from the controller, convert the response to a proper JSON format and provide the corresponding status code for the request.

Up to this point, Etqan has created a fully RESTful API system, however, this system doesn’t have any access control logic yet. Access control should for such a system should be editable, where the user can change roles and permissions at any point, therefore the access control should be defined completely outside the code and should be dynamic.

3.6.The Design of Grant library

Grant which is a library that was created specifically for this system it is a Role-based Access Control (RBAC) Library for Haxe programing language, that allows you to manage all your RBAC in an easy and flexible way, The idea of Grant is that all RBAC should be kept outside the code, it is maintained in a JSON file. RBAC can be easily changed by only modifying the JSON data. However, Grant allows you to have your RBAC logic in the code as well.

Grant defines five entities in its system:

1. Resource: which refers to any data to be accessed that were created by the user, it can be in a textual or multimedia format, such as a user comment, an image, or a tag in an image.
2. Role: a user-defined entity, the user decides on what roles to create, gives them names, and attaches to them policies.
3. User: the user who would like to access the data, he should have a role to access the data, if the user has no role, a special role called 'public guest' is assigned to him/her
4. Policy: a policy that allows resources to be created, read, updated, or deleted by the user
5. Condition: a condition that is set on the policy, such as expiration dates or number of times to access the resource.

Grant is an access control library has many features which are:

- All RBAC logic maintained in JSON data format, the data can be saved in a SQL database, a no-SQL database or even as JSON text files on the server. The JSON format is friendly and easy to build and structure.

- All RBAC logic is maintained outside the application code, this enable a clear separation between RBAC and application code, RBAC can be modified without the need to access the application code at all. Decoupling RBAC roles and permissions from resources so they can be re-used
- Grant library is responsible to handle all access control and permissions on resources, and it communicates directly with the database engine if resources are stored in a database. The use of Grant should be enough to secure the resources and provide all possible RBAC, however it also possible to use database security and checking features for an extra layer of security.
- Fine-grained access control to any record or resource is provided through the use of conditions, conditions are constructed by the end-users and then applied to RBAC, if another user is granted a role with a condition, he/she cannot access the resource unless the condition is met, the condition can be based on the resource itself or the user who is trying to access the resource, for example if the user is a friend to the resource owner then a role with appropriate permission can be assigned the user automatically.
- Grant system makes it possible to add more than one policy per role to the same resource or to the same data table, making it possible for the same role to access the resource differently based on the context it is accessing the resource.
- Inheritance of roles is an important feature that is supported by Grant, it helps managing the RBAC easily, Grant offers flexibility in managing inheritance where multiple inheritance is supported, however this may cause conflict of access when inheriting from two different roles that have contradicting access on the same resource. Grant deals with conflicts based on priority of orders and based on context, if role C has a permission to access resource R based on two

permissions that are inherited from Role B and A, if role C inherits from B after A, then the permission on B will override the permission on A.

- RBAC can also be assigned temporarily or altered automatically by the actions taken, for example it is possible to add limits to policy e.g., how many times to read a resource, or to revoke a policy once it was used.

The system itself consists of two main sub-systems, the first sub-system which is called Grant-UI it allows resource owners to create roles and assign policies on these resources, the second sub-system is called Grant-Command which allows other applications to manage access on resources.

The Grant-UI sub-system consists of three components as shown in figure 3.11 which illustrates how the sub-system works to generate the final RBAC policies.

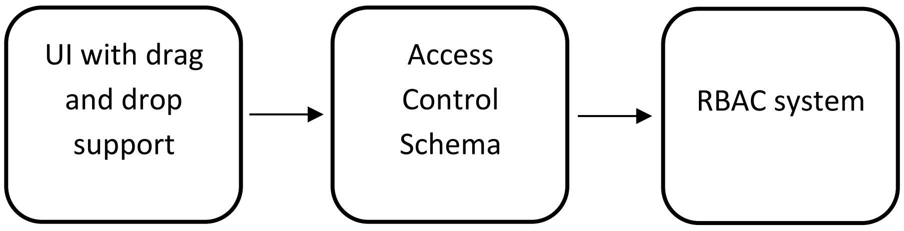


Figure 3.11: Grant system architecture.

The UI system consists of a drag and drop interface where the users create roles and policies on resources, then assign policies to these roles, and finally, assign roles to users. The UI is a user-friendly interface to generate the JSON data which defines the actual Access Control Schema that is accessed by the application, the JSON data is read by the Grant-Command sub-system when a request to access the resource is made. The Grant-Command workflow is illustrated in Figure 3.12.

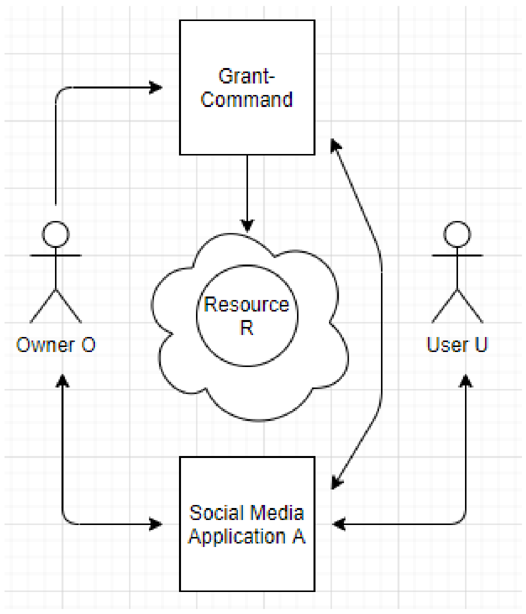


Figure 3.12: Grand-Command Workflow

3.7.The output of the Etqan system

The final output of the Etqan system is a fully RESTFUL API, this API will contain many things include:

- SQL data base, which include primary keys, foreign keys, relationships, database validation and data tables.
- URL routing, which includes (GET, POST, PATCH, DELETE, PUT methods), relational paths, status codes, partial response, search paths, and pagination.
- Access Control.
- server validation, this includes check correct data type, check correct size, regular expressions, and limits.
- Mock Data.
- Caching.

Figure 3.13 shows the RESTFUL API components.

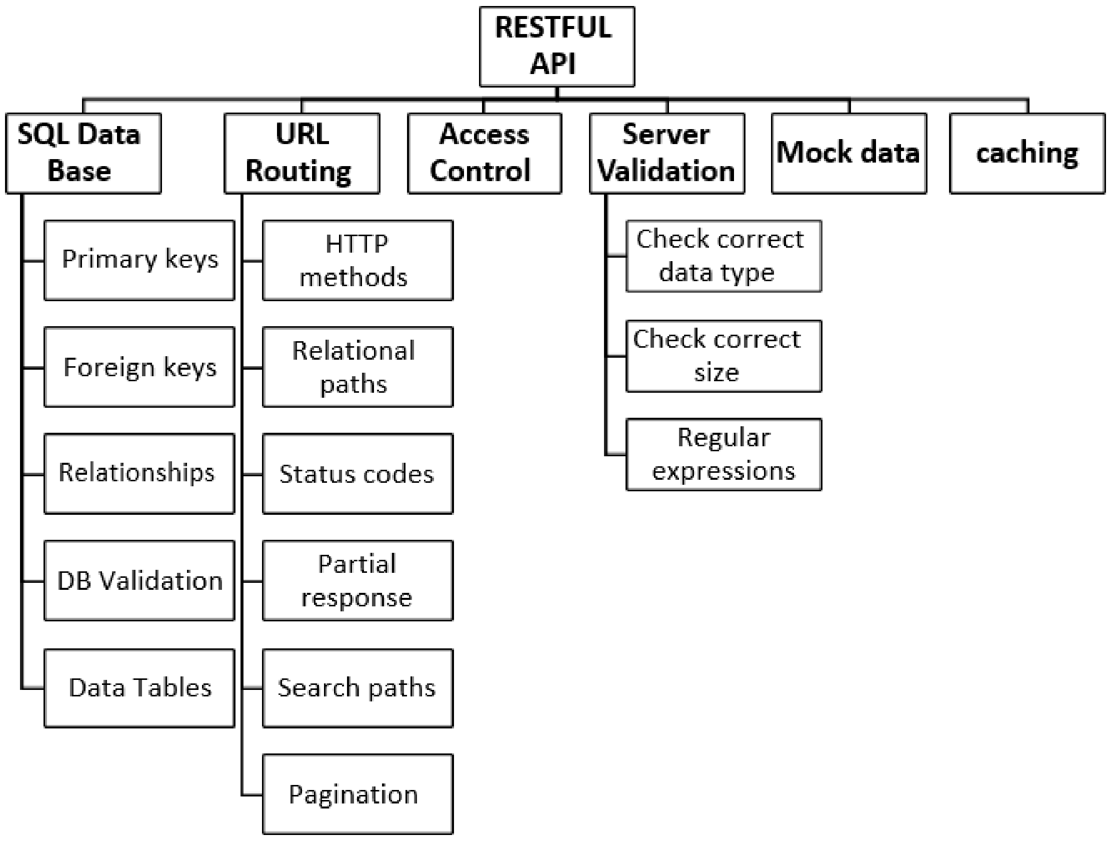


Figure 3.13: RESTFUL API components

We used many steps to creating a REST Web service, these steps are:

1. Identify which resources should be published.
2. Design resources URLs.
3. Determine resource relationships.
4. Decide which representations will be available.
5. Define which methods are available for each resource with a description of its effect.
6. List the possible responses (http codes and result).
7. Document each resource.
8. Discover services.

3.8. System process flow

Figure 3.14 shows the flow of the processes running in the system , the Etqan system is triggered when a POST request is submitted with the JSON data that follows Etqan JSON Schema when receiving this request, Etqan handles it to the JSON parser and validator when the JSON is passed and successfully parse, a new project will be set up with all the required configurations such as Name, date of creation, version, package, owner and super admin credentials, then Etqan system will start identifying classes and their relationships, including their attributes and validations rules, after that the models will be built, the models will form the base for each class an d will be corresponded with a physical database table in an SQL database because the models are now ready, it is possible to create the database layer which will be responsible in creating the database and the tables using Haxe ORM (Object Relational Mapping). The Haxe ORM system provides a high-level abstraction of SQL statements, therefore the Etqan system doesn’t need to deal with any SQL statement. After that Etqan will start building the controllers classes and add the access control rules as the controller will work directly between models and the API layer which deals with external requests. the API classes will be created after that, there will be one controller and one API class will be created per model, the API classes will contain all the routing logic, another pass to review relationship is needed to ensure all relationships are met successfully, especially the many-to-many relationships. This pass will also handle data integrity and make sure the relationship constraints meet the requirements. Now the project will be ready which is written in Haxe code, Etqan will save the project in a directory, supporting files and data will be added to the project at this stage, next step will involve Etqan installing all Haxe dependencies for the project, then compile the project to generate the final PHP project, this PHP project is ready for deployment and Etqan can deploy it to a live server or allow the user to download it a zipped file, the user will need to run the API that allows building the database using his super-admin credentials, then the system will be live online.

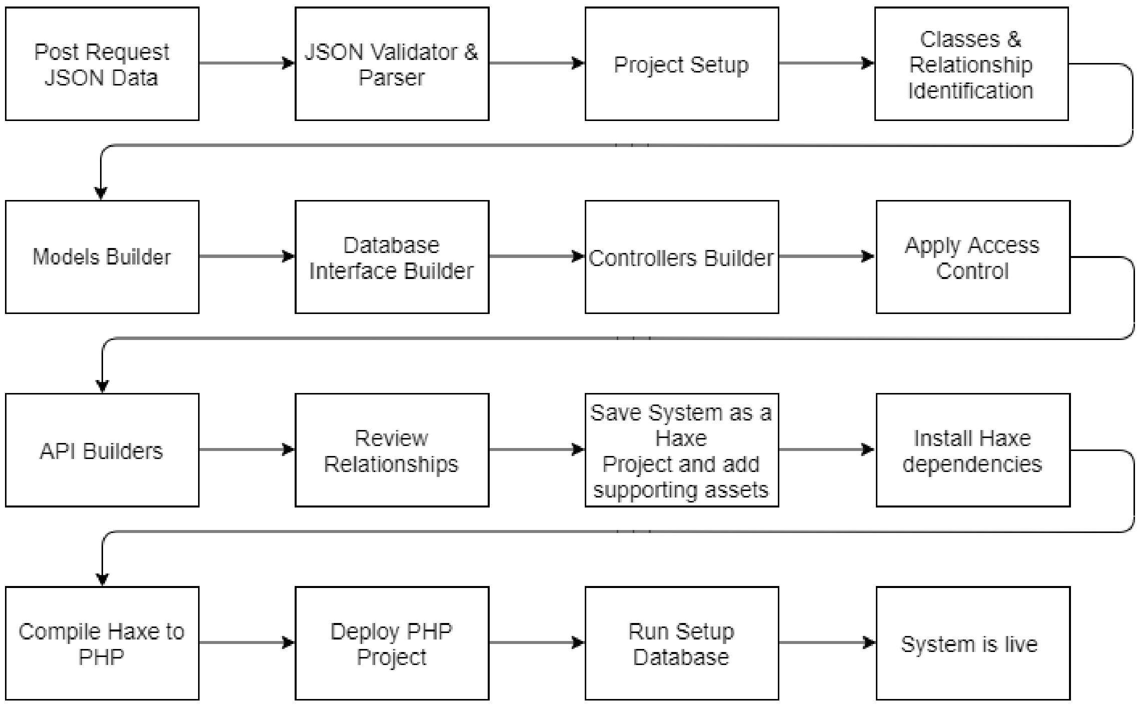


Figure 3.14: system process flow

3.9.UML For Etqan System

Figure 3.15 shows the UML design of the system, Etqan system has 5 classes, the API Maker class is the class that is responsible for taking responsibility to run Etqan workflow as shown in figure 3.15, and performing any intermediary stage between the main stages such as setting the project and reviewing relationships. Each of the remaining classes will have a specific role within the workflow, as the name suggests Table class maker is responsible for creating models, the API Class Maker for creating API classes, and the Controller Class Maker will be responsible for creating controllers. The Root Path Class Maker is responsible for creating an entry point for external requests to perform the first round of routing, then the request will be handed to one of the API classes afterward.

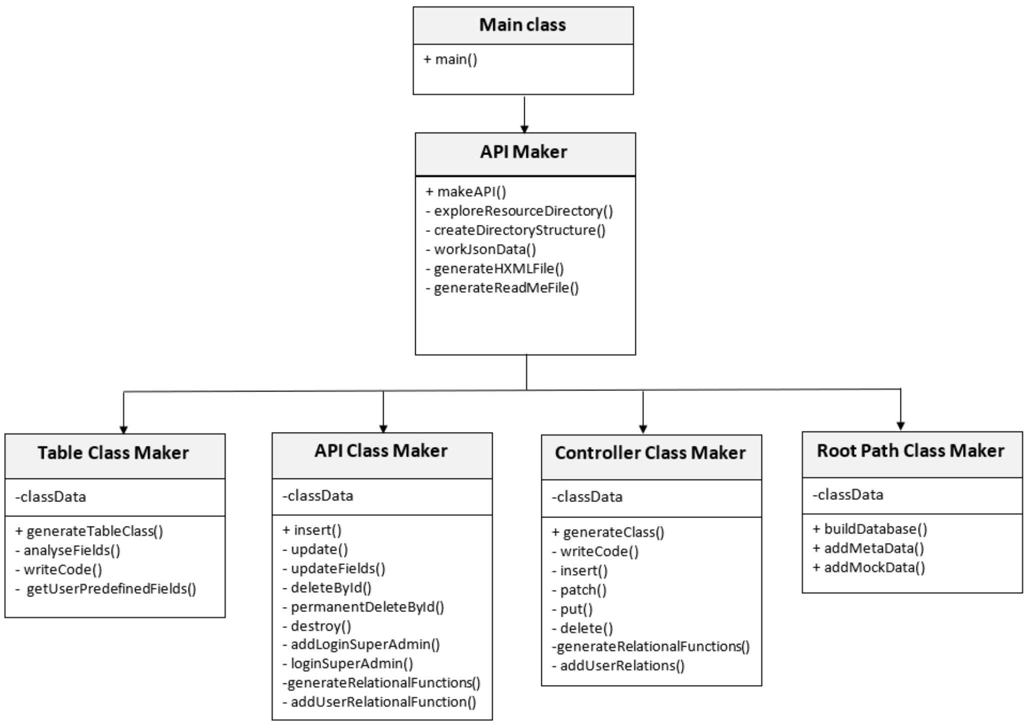


Figure 3.15: Etqan UML diagram

3.10. UML for generated system

Figure 3.16 shows the UML design of the generated API system, the API system has 8 classes.

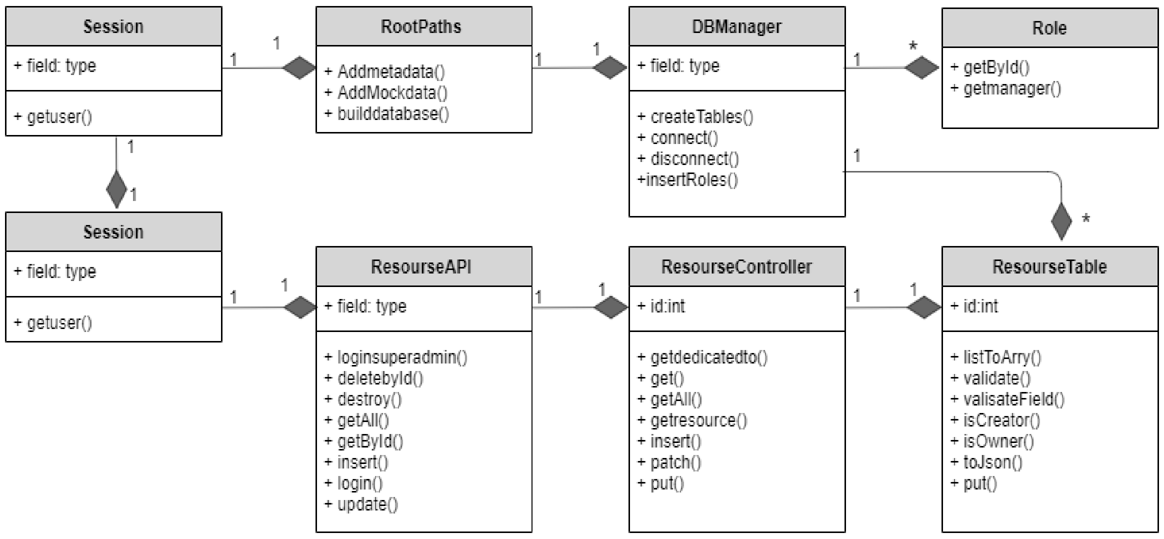


Figure 3.16: UML for generated system

The server class is responsible for running the whole API system components and synchronize the work among them, it is also the starting point of the system, the second class is the root path class which decides where each request should be transferred to its corresponding API class, this API class works with controllers, the controller class is the only class that talks directly to the models and it contains several functions to dispatch actions on the model as well as verifying and validating the data. The last class is the Database manager to handle physical database requests.

CHAPTER 4

4. SYSTEM IMPLEMENTATION

In this chapter, we will describe how Etqan generates the APIs in section 4.1, section 4.2. We will describe developing Etqan system then in section 4.3 we will describe Etqan code architecture after that we will describe the grant library permission, limits, and conditions in section 4.4, finally in section 4.5 we will describe how to run Etqan.

4.1. How Etqan generates APIs

Etqan system is written in Haxe language, Haxe consists of a high-level, open-source programming language and a compiler. It allows the compilation of programs, written using an ECMAScript-oriented syntax, to multiple target languages. Employing proper abstraction, it is possible to maintain a single codebase that compiles to multiple targets. Etqan generates API systems in Haxe, however, the current implementation of Etqan supports only PHP v7.4 with plans to add support for Node.JS in the future.

Etqan system accepts a path to an Etqan JSON file generated by Etqan UI, the filename is passed as a parameter through POST request body, then the Etqan service starts to scan the file contents to generate another Haxe API system based on the specifications inside the JSON file, the generated API system uses also written in Haxe, it uses some Haxe libraries including Haxe record-macro library which support object-relational mapping (ORM) which the Etqan system uses to create the corresponding Database and generate the database constraints. This API system still needs to be compiled and to generate the final PHP API system figure 4.1 shows these steps.

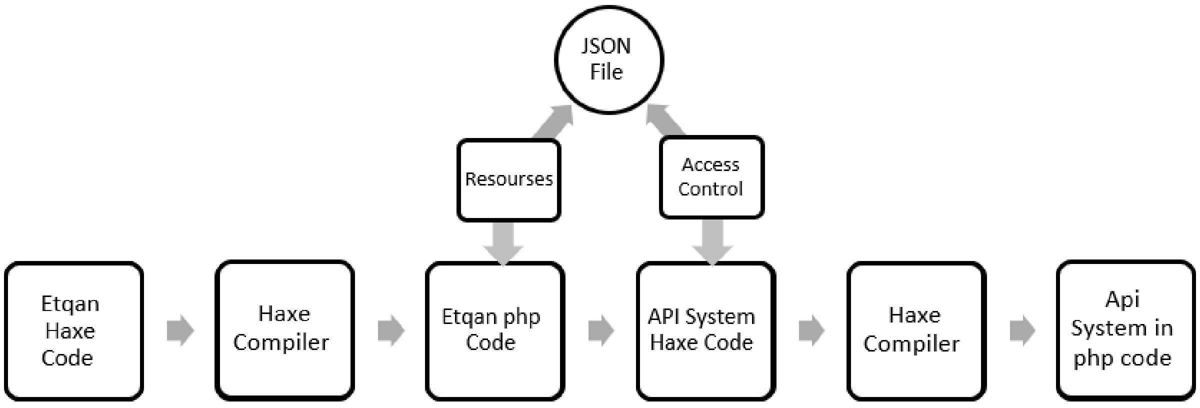


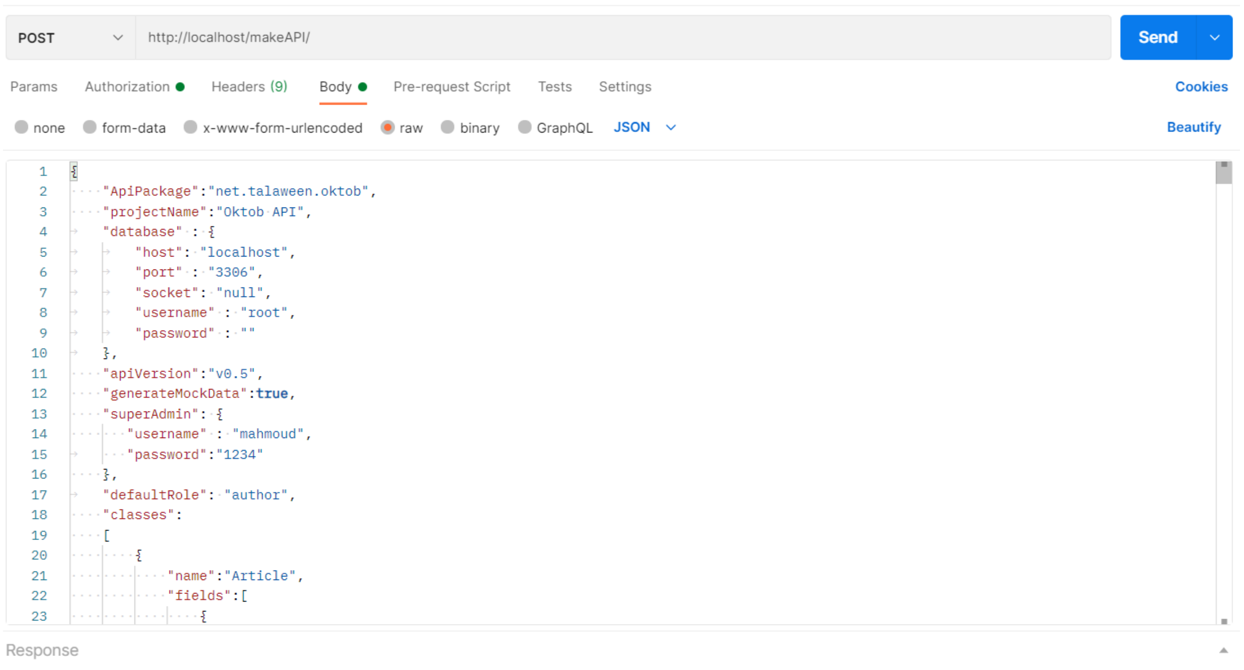
Figure 4.1: Generate the final PHP API system

The Etqan system is an API generator, based on the available JSON files from the Lava system, the Etqan system starts working by receiving an HTTP request. This request includes a JSON file for the API system we'll create, which is included in the body. To begin, the client should use the post method to request the creation of the API at the following URL:

```
http://localhost/makeAPI/
```

To clarify how the system works let's take an example:

Suppose that we would like to develop an API system for a blog application, in which articles, authors, and comments are the main resources, the JSON file for this blog will be created through Lava, then passed to Etqan to create the API for this blog, the request will be sent as in the Figure 4.2:


POST http://localhost/makeAPI/

```
{
    "ApiPackage":"net.talaween.oktob",
    "projectName":"Oktob API",
    "database" : {
        "host": "localhost",
        "port" : "3306",
        "socket": "null",
        "username" : "root",
        "password" : ""
    },
    "apiVersion":"v0.5",
    "generateMockData":true,
    "superAdmin": {
        "username" : "mahmoud",
        "password":"1234"
    },
    "defaultRole": "author",
    "classes":
    [
        {
            "name":"Article",
            "fields":[
                {
```

Figure 4.2: HTTP request to generate API

Etqan starts reading the information's from JSON in the body for the HTTP request and storing the information it contains.

Then Etqan system begins with the creation of the directory structure for the targeted API system, firstly, Etqan needs to generate an HTML file and save it in the root directory, a HTML file is a Haxe file that contains all the information Haxe compiler needs to compile Haxe code to a targeted platform. Etqan creates the source folder that will contain the package path and all source files include APIs, controllers, and models, Etqan will also run through all classes to add a relationships function.

4.2. Developing Etqan System

An Agile methodology was used to develop the Etqan system, the backlog of the project contains all required tasks that are classified according to their importance. Git was used as version control with Bitbucket used as the hosting platform for the repository see links in Appendix A.

Visual Studio Code is used as the primary code editor with Haxe language 3.4 installed, and a Haxe plugin for visual studio is also installed.

The source folder of the Etqan contains three main sub-folders:

- JSON Examples: which contains example of JSON files used for testing purposes and demonstrations
- API maker: which contains the actual source files of Etqan
- Resources: which contains assets and files that are needed by the generated system to support functionality of the API system, these are rarely modified and contains files such as a list of available languages, gender, operating system in addition to Haxe source files that will always be the same for all API projects.

Etqan makes use of Haxe powerful string template system, which allows injecting values inside a string easily, a Haxe string template starts and ends with a single quote, to inject a variable the dollar sign with curly braces are used as in this example:

```
'Number of files generated are ${fileCount}';
```

The main.hx file is the system entry point, which passes a valid POST request with a valid file name to APIMaker.hx, which is the class as explained earlier, responsible for overall workflow and takes the output of one operation to another one when starting it. The system also contains a file called SourecTemplates which consists of several code pattern templates that Etqan uses in many places. A Utility class also exists that contains several utility functions used by Etqan to facilitate its operation, such as dealing with

Dates and Times values, finding relational types, identifying if any keyword is used in the wrong place, converting between Haxe types and search functions.

The system has a Settings.hx class which is responsible to handle all current settings, such as where to save the output, author name, API version, database credentials, super-admin credentials, constants used to identify patterns, and code templates.

The rest of the code files are corresponding to the main system classes as explained in the previous chapter which handles one major stage of generating the API, these files are:

TableClassMaker.hx, ControllerClassMaker.hx, DBManger.hx, APIClassMaker.

4.3. Etqan Code Architecture

Figure 4.3 shows the code architecture of Etqan and how each file is related in the final generated API system, each file contributes to the final system:

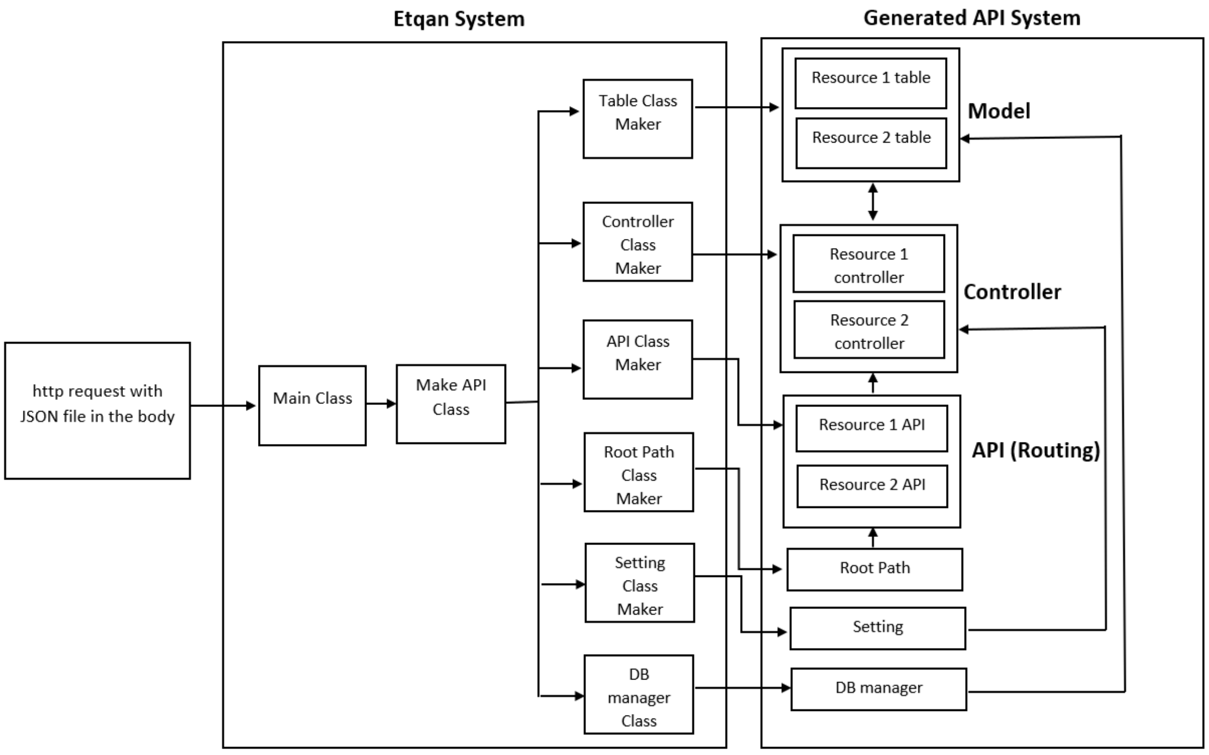


Figure 4.3: The code architecture of Etqan

When the Etqan system receives a request, the main class from the Etqan system routes it and defines the path through which the request should be communicated. It also validates and extracts the JSON schema and then transfers it to the second class, which is known as the make API class; this class is in charge of generating the final API by organizing work across the many classes of the Etqan system, such as: table class maker, controller class maker, API class maker, root path class maker, setting class maker, and data base manager maker class.

Each class in the Etqan system is in charge of creating a certain class in the Generated API system:

- The Etqan table class maker generates the resource table class in the Generated API system.
- The Etqan controller class maker generates the resource controller class in the Generated API system.
- The Etqan API class maker generates the resource API class in the Generated API system.
- The Etqan root path class maker generates the root path class in the Generated API system.
- The Etqan setting class maker generates the setting class in the Generated API system.
- The Etqan database manager class generates the data base manager class in the Generated API system.

Generated system Architecture consists of:

- The model, which represents the tables in the system.
- The controller, which works with models, has the primary function of implementing actions on the model as well as verifying and validating the data.
- API (routing) This class's major role is to work with external requests, validate the data in the requests, and protect the system from any threats or malicious attack or malicious request.
- Root path: to determine where each request should be transferred and forward the request to his API.
- Setting class which includes the global setting for the generated API system such as the name, the version, the name of the author.

- Database manager: The main function of this class is to handle physical database requests.

4.4. The implementation of Grant library

Grant is a Role-based Access Control (RBAC) Library for Haxe, created for this project, the idea of Grant is to allow developers to manage all their RBAC in an easy and flexible way. Grant is a complete system with ability to create roles, permissions, conditions, and relationships, it also allows creating special kind of access control policies such as number of accesses, expiration date or conditional access.

Grant has one important rule, that all RBAC should be kept outside the code, where it is maintained in a JSON file. RBAC can be easily changed by only modifying the JSON data. however, Grant allows you to have your RBAC logic in the code as well. By having this rule, it is easy to change access control without the need to change the base code or recompile the project.

Currently, the library works with Haxe version 3.2 and only generates PHP v7.4 and supports only MySQL, in a future release, it will be available for Node.JS, Python, JAVA, and C#.

4.1.1. JSON file Schema

All access control data are stored in a JSON file or an anonymous structure in the code, if it is in a JSON file then it is the developer responsibility to prevent access or modifications to the file.

The top-level object of the schema, which has only one property called "accesscontrol" which is an array of roles.

```
{"accesscontrol": Array<Roles>}
```

The Role object has the following properties:

- Role: name of the role.
- Inherits: an optional field which is the name of the role to extend/inherit from, For instance, the user can make use of the inheritance property to define the base role whom permissions are to be inherited. By this property we apply the meaning of role based access control, where permissions could be inherited from another base-role. By doing so, we get all the permissions of that role it inherits, in addition to the new permissions added to the new role.
- Grant: an array of Resource objects which the user role can access.

```
{"role": String, "inherits":String, "grant":Array<Resource>}
```

The Resource object has the following structure and fields:

```
{"resource": String, "policies":Array<Policy>}
```

- Resource: the name of the resource.
- Policies: an array of policy objects to be applied on the resource.

The Policy object has the following structure and fields:

```
{"action": String, "fields":String, "records":String,
"limit":Limit}
```

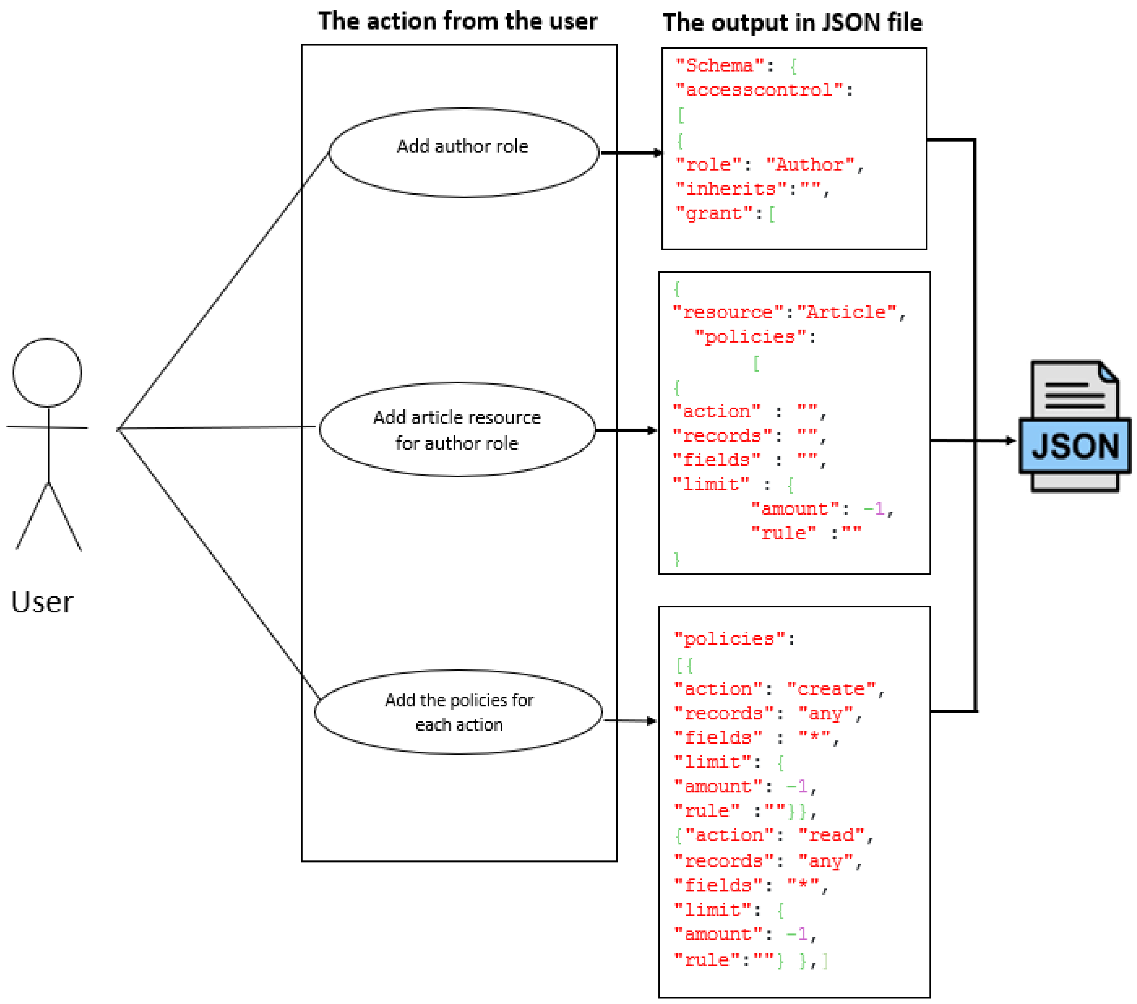
- Action: one of the following create/read/update/delete.
- Fields: the fields the role can access from this resource, the fields should be separted by comma e.g. " `title, maintext, publisheddate` ". You can

also specify all fields "" *and exclude some fields with "!", e.g.* `",`

`!Publisheddate"`

- Records: the condition(s) that allow the role to access this resource, see below for explanations on conditions.
- Limit: is a limit object to count how many times a role can perform the specified action on the resource.

For Example, adding resources, adding roles, and adding policies all have their own particular structures within the JSON file, as seen in Figure 4.4. The user in the illustration first adds the author role, followed by the article resource, before adding policies for each resource.



```
"Schema": {
"accesscontrol":
[
{
"role": "Author",
"inherits":"",
"grant":[
```

```
{
"resource":"Article",
  "policies":
        [
{
"action"  :  "",
"records":  "",
"fields"  :  "",
"limit"  :  {
        "amount": -1,
        "rule"  :""
}
```

```
"policies":
[{
"action": "create",
"records": "any",
"fields"  : "*",
"limit": {
"amount": -1,
"rule"  :""}},
{"action": "read",
"records": "any",
"fields": "*",
"limit": {
"amount": -1,
"rule":""} },]
```

Figure 4.4: JSON file schema

A developer can apply more than one policy for the same rule on the same resource, with each Policy gives the user a different access to a subset of records, for example one Policy allows user to read all the fields of his own articles, another policy to read a subset of fields for articles he does not own. The order how you add policies on the same resource is important, If you have more than one policy on the same resource, Grant always execute the first policy that give the user access to the record.

The Limit object has the following structure and fields:

```
{"amount": Int, "rule":String}
```

- Amount: the number of times the role can perform the action on the resource, use -1 for unlimited, 0 can be used to ban.
- Rule: the condition needs to count the limit, see below for explanations on conditions.

A developer can specify more than one policy on the same resource for the same role, so that the role can access different records in different manners, the record property of a policy object allows a developer to decide which rows/records in the table the user can access. For example an author role will allow the user to read any article and see its title, text body and published date, and another read policy that is applied to his own created articles where he will be able to see title, text body, published date and notes.

4.1.2. Grant permission

A Developer can control sub-object permission (currently is restricted to one level) by Grant.

For example, if a developer has a resource called Comment which also has an article and user resources associated with it, then the developer can ask Grant to apply the available policies for these resources that are available for the current user by identifying the sub-object with the special character '^' operator as follow:

```
{
    "action" : "read",
    "records": "any",
    "fields" : "commentText, publishedDate, author^User,
     article^Article",
    "limit" : {
        "amount": -1,
        "rule" :""
    }
},
```

Grant will look for what User policy is available for the current user and apply it on the author field, and the same for article, if Grant cannot find any policies or the current policy does not allow access the resource then that sub-object will not appear.

If a developer does not apply the "^" operator then no access control policy will be applied to the sub-object and it will be returned as if the user is allowed to access the sub-object as in the current policy.

4.1.3. Grant Conditions

The conditions are very important part of how Grant works, it allows developers to specify how to access a specific record, it is in fact allowing to filter which records/rows of a table a user may access and how it should be accessed.

Conditions can be specified on two fields:

1. Records field of a Policy object.
2. Rule field of a Limit object.

A Developer can specify values directly from the user object or the resource object, add a select statement to the condition or combine both.

There are few simple rules on how to write conditions to ensure that Grant can get the correct answer:

If Etqan is checking any values that is on User or the actual resource object then the developer should use the following syntax, E.g. Check if the user is the actual author of the article.

```
"records" : "$resource.auhthorId=$user.Id",
```

Note that Grant used the word "user" to refer to the actual current user who is currently trying to access the resource, and the resource word to refer the current resource being accessed.

A developer can also add static values when necessary:

```
"records" : "$resource.price >= 20/i",
```

Note that adding a tailing flag to indicate the data type of the value:

```
/i for integer e.g. 20/i.
```

```
/f for float e.g. 13.5/f.
/b for boolean e.g. True/b.
/d for date e.g. 2019-07-14/d.
```

Otherwise, the value will be treated as a string.

The comparison operator can be one of the following (>, >=, <, <=, ==, =, !=), where = or == will be treated the same.

If checking values from any other tables or objects, then the developer has to use an actual SQL select statement, e.g. check if the user can access a picture which only allowed for users who were tagged in it:

```
"records" : "Select count(*) From tags where tags.userId =
$user.id And tags.pictureId = $resource.id",
```

If the length of result for this SQL is more than 1, it will be evaluated as true otherwise false.

These two examples can be combined using either | & operators:

```
"records" : "$resource.auhthorId=$user.Id | Select * From tags
where tags.userId = $user.id And tags.pictureId = $resource.id",
```

4.1.4. Grant limits

The Limit object provides a condition to count how many times a policy can be valid before it expires, the amount property specifies the maximum number to execute the policy, while in the "rule" property, a developer should use a SQL SELECT statement that counts the number of times the user role has accessed the object, for example allowing a user to only create maximum 5 articles:

```
"amount" : "5",
"rule" : "SELECT count(*) From Articles where
Articles.authorId=$user.id"
```

In this example if the SQL statement returns a result less than 5 the user will be able to access and to perform the action on the resource.

4.5. How to Run Etqan

Apache Server is an open-source web server that can run in Windows, linux or Mac. Under windows there are different bundles that provide Apache server which also includes some other related applications such as MySQL server and PhPMyAdmin. WAMP server is such an example.

To run the Etqan system Apache server and MySQL server should be running.

Haxe system and Haxe libraries

First install Haxe v3.4.7

This is a Haxe project, which is needed to be compiled to PHP using the following command:

```
haxe build.hxml
```

This will generate an Etqan system in PHP in the path specified inside the build.hxml file in line3

Etqan system needs one folder to be placed inside its root folder (e.g. /makeapi folder inside www folder of WAMP server).

This folder is called resources, and should be copied from this source into /makeapi folder.

In the www root directory create a new folder and call it download

Currently Etqan API system only runs with Haxe v3.4.7 and the specific older libraries called tink_web and tink_core

Install version 0.1.5 for tink_web using the command inside your terminal

```
Haxelib install tink_web 0.1.5
```

Then install tink_core v1.22.0 using the command

```
Haxelib install tink_core 1.22.0
```

You will need to set the active version of tink_core to 1.22.0 using the command

```
Haxelib set tink_core 1.22.0
```

You will need to add Grant library as external library from git using the command:

```
Haxelib git Grant https://github.com/Talaween/Grant.git
```

Install record_macros lib with command:

```
haxelib install record-macros
```

install random lib with command:

```
haxelib install random
```

install http-status lib with command:

```
haxelib install http-status
```

Now Etqan is installed and ready to work.

4.6. Generated Systems Architecture

The process of creating a software interface that exposes backend data and application capabilities for usage in new applications is referred to as API architecture, which is great for microservices.

Providing a decent developer portal or a high-performance gateway isn't enough when it comes to effective API management. API management plays several critical responsibilities in the modern digital organization, with three Layers of API Architecture, these layers are shown in figure 4.4:

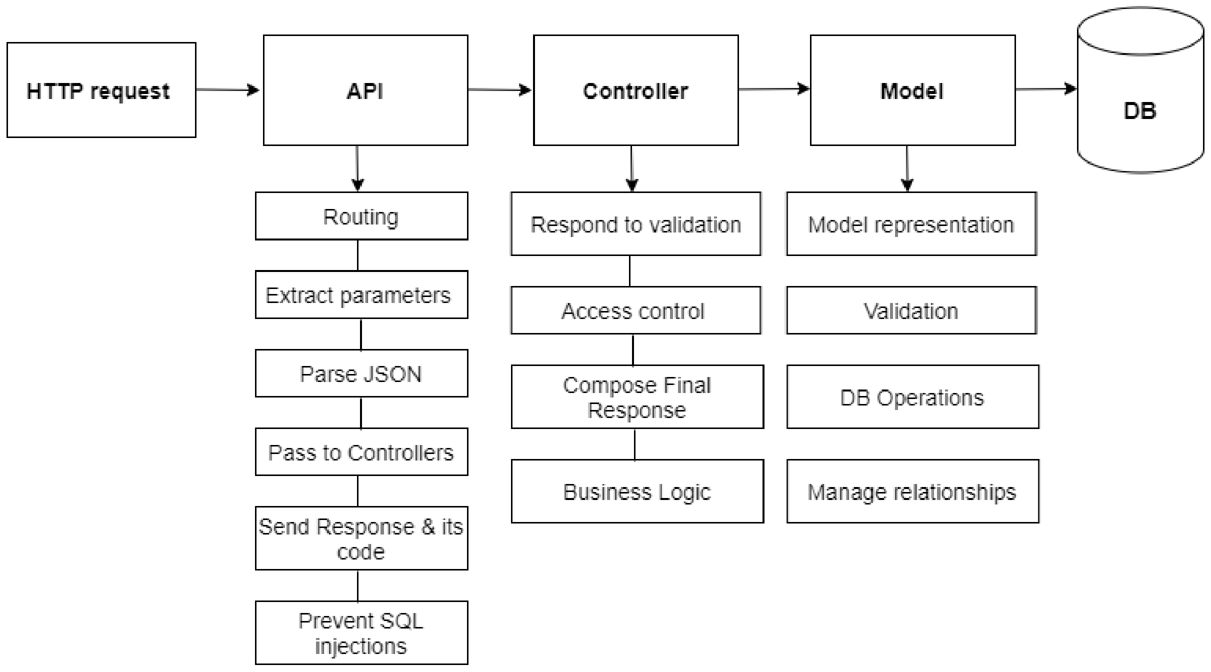


Figure 4.5: Generated API architecture

Figure 4.4 shows the generated API architecture, it consists of three layers, first layer is the API which is responsible to communicate directly with external http requests, extract parameters, parse json file, guard the application against SQL injections and finally passes data to the next layer which are the controllers, the controller layer is responsible to respond to validations results, perform all business logic, access control logic, and compose the final response, the controllers are communicating with both the

API layers and the model layer, which represents the data, perform validations, manage relationships and performs all database operations.

The chosen architecture divides the operations into several layers, which is easy to configure, build and maintain. The architecture is scalable, which means any new models can be added easily to the system without affecting the existing code base, and if there are thousands of models it still can manage to handle them efficiently. As this architecture is repeated for each resource, it has a well-defined code design pattern that can be adapted in Etqan code templates.

The generated system has the following features:

For each resource:

- GET one instance of the resource by using its unique id

  /resource/id

  e.g. GET /books/20 ➔ gets the book with id 20

- GET all possible resources with pagination support

  /resource?page=#&limit=#

  GET /books?page=1&limit=20 ➔ get the first 20 books this user can access

- GET relational resources

  /resource/id/relationalResource/id

  e.g. GET /books/20/author/15 ➔ gets author of book with id 20 who has id 15

  /resource/id/relationalResource

  e.g. GET /books/20/author ➔ get all authors of book 20

- POST to create a new resource

  /resource

  e.g. POST /books ➔ insert a new book

- PUT to fully update a resource

  /resource/id

  e.g. PUT /books/20 ➔ fully update book with id 20

- PATCH to partially update a resource

  /resource/id

  e.g. PATCH /books/20 ➔ partially update book with id 20, update only its title

- DELETE a resource

  /resource/id

  e.g., DELETE /books/20 ➔ delete book with id 20

the generated system will have data validation all the way and will generate the proper response code based on the request:

Table 4-1: Response code table

Code	The proper response code based on the request
Code 200	retrieved data correctly.
Code 201	inserted a new resource successfully.
Code 204	the server returned an empty json object.
Code 206	partial response to the client, the server only gave part of the requested resources.
Code 400	bad request, validation error.
Code 401	authorized, user has no permissions to access the requested data.
Code 402	payment required; the data requested needs payments to be obtained.

Code 404	method or action not found.
Code 405	method not allowed on the current resource.
Code 408	request timeout, the request took too long time to be processed.
Code 500	a server error, for example database server is down.

CHAPTER 5

5. SYSTEM VALIDATION AND EVALUATION

Software system reliability is a critical performance metric to an organization, which has four factors’ people, hardware, software, and data. the reliability of software is a joint responsibility of information system professionals. Regardless of the types of software involved, the reliability of software must be achieved through a continuous effort of planning,

Studies have shown that most of the system errors detected during the testing phase originate in the early analysis phase (Boar, 1984). Therefore, software testing should start early in the system development process.

Software testing should not be confused with software debugging. Software testing is a process of finding "unknown" errors whereas software debugging is a process of removing "known" errors. In general, the objective of testing is to discover errors in a system and to verify that this software does what it is supposed to do, in addition, to see if it does not do what it should not. Software Testing can be considered as a destructive process and debugging as a constructive one. However, testing and debugging should go hand-in-hand and be performed one right after another.

Once the program development (PD) phase takes place, manual testing should be performed before and after a program is subjected to a computer-based test. The purpose of such manual testing is to ensure that the programs being developed in this phase confirm the System Specifications. After the program development (PD) phase, manual testing is performed to ensure that the current phase documents were all consistent with the end products of the prior phases along with the feedback from the end-user (Li, 1990).

5.1.Manual Testing Plan

To test Etqan system, a test plan was created, this test plan involves creating different applications specifications from simple to more complex ones, to make testing consistent, a development system called Oktob was used, this system is a blogging system, and has different variations, the link of this JSON file in Appendix A.

- **Oktob Level 1**

A simple blogging system that consists of two resources: User and article, and has three roles: reader, author, and Guest, table 5-1 shows the plan for this level.

- **Oktob Level 2**

In this level a comment resource is added, table 5-2 shows the plan for this level.

- **Oktob Level 3**

In this level a reading list resource is added, a new role: moderator is added, table 5-3 shows the plan for this level.

- **Oktob Level 4**

In this level a Like resource is added, a new role Premium author is added, the ability to publish and unpublish articles is added, also type of articles is added Premium articles and free articles, table 5-4 shows the plan for this level.

A set of tasks needed to be completed to verify that the generated systems work as expected.

Oktob Level 1 Testing Plan

Table 5-1: level one testing plan

Number of the test	**Tasks**
O1T1	A Guest Can register
O1T2	Users can add a new article
O1T3	Non-register users can't add an article
O1T4	User can update his own article
O1T5	A user cannot update other user's articles
O1T6	Register user cannot register again
O1T7	A user can delete his own article
O1T8	A user can't delete other user's articles
O1T9	A user can read all information about his article
O1T10	A guest can't read all the fields for the article
O1T11	A user can't create an account for another user
O1T12	A user can update his own account
O1T13	A user can delete his own account

Oktob Level 2 Testing Plan

Table 5-2: Level two testing plan

Number of the test	**Tasks**
O2T1	A user can add a comment on his articles.
O2T2	A user can add a comment on other user's articles.
O2T3	Non-register can' add comment.
O2T4	A user can delete his own comments.
O2T5	A user can update his own comments.
O2T6	A user can read any comment.
O2T7	A user cannot update another user's comment.
O2T8	A user cannot delete another user's comment.

Oktob Level 3 Testing Plan

Table 5-3: Level three testing plan

Number of the test	Tasks
O3T1	Register user can add an article to his reading list.
O3T2	A user can delete an article from his reading list.
O3T3	A user can read his reading list.
O3T4	A user can't read other users reading list.
O3T5	Non-register users can't add a reading list.
O3T6	A moderator can add a user.
O3T7	A moderator can't delete users.
O3T8	A moderator can add a new article.
O3T9	A moderator can add a comment.
O3T10	A moderator read all resources.

Oktob Level 4 Testing Plan

Table 5-4: Level four testing plan

Number of the test	**Tasks**
O4T1	Register user can add like.
O4T2	A user can delete his own like.
O4T3	A user can see all likes on the article.
O4T4	Author can publish his article.
O4T5	Author can unpublish his article.
O4T7	Premium author can add premium article.
O4T8	Author can add free article.
O4T9	Premium author can update his premium article.
O4T10	Register user can read the free article.
O4T11	Non-register user can't read the free article.
O4T12	Non-register user can't read the premium article.

5.2.Manual testing Procedures

The JSON data has been created for each variation of Oktob system using Notepad++, then it is manually verified and tested to make sure JSON data is valid and according to the schema.

The following steps were performed in each test case:

1- The full API endpoint is written.

2- all required parameters are prepared including authentication, POST parameters, path parameters or query strings.

3- the expected behavior is written down.

4- a test then takes place.

5- the result is registered.

The testing procedure involves the following steps after generating a new system from a test schema:

- Make a POST request to

```
http://localhost/makeAPI/
```

Before sending the request, we put the JSON data in the payload and make sure the encoding type is JSON.

- You will be given the generated path to the new system in the response to your POST request
- Use that path to perform all subsequent operations e.g.:

```
/sys/OktobAPI20210516222142/oktob/raneen/build
```

- The path to build the database will be in:

```
localhost/sys/{project directory name}/{package name last
part}/api/{api version}/{superadmin username}/build
```

- To add a user for example the path to the API should be:

  ```
  POST => localhost/sys/{project directory name}/{package name
  last part}/api/{api version}/user
  ```

- Then add a basic auth as per super admin username and password in your json data for that request to build the database.

After these steps, each level was tested thoroughly, the results of each test is documented in section 5.3.

All tests took place in a Laptop Computer with the following specs: That runs Windows 10 professional, Apache Server, PHP v7.4 and MySQL locally.

The website https://jsonformatter.org/json-parser has been used as a method to parse and validate json schema.

5.3.Manual Testing Results

Oktob Level 1 Testing

Table 5-5: Level one testing results

Number of the test	Tasks	Expected Result	Actual Result	Pass or Fail
O1T1	A Guest Can register	Create a user account in the database	The account has been added to the database	Pass
O1T2	Users can add a new article	The article must be added to the database	The article has been added to the database	Pass
O1T3	Non-register users can't add an article	The system should refuse to add the article	The system sent the following message: "unauthorize attempt to insert data"	Pass
O1T4	User can update his own article	The article should be update	The article updated	Pass
O1T5	A user cannot update other user's articles	The system should refuse to update the article	The system sent the following message: "unauthorize attempt to Update data"	Pass

O1T6	Register user cannot register again	The system should refuse to register the user	The system sent the following message: "value already exist: { username : raneeen }"	Pass
O1T7	A user can delete his own article	The article should be deleted from the database	The article was deleted from the database.	Pass
O1T8	A user can't delete other user's articles	The system should refuse to delete the article	The system sent the following message: "unauthorize attempt to delete data"	Pass
O1T9	A user can read all information about his article	The system should send the information for the article	The system Sent the information for the article	Pass
O1T10	A guest can't read all the fields for the article	The system should send just the global information for the article	The system Sent the title and the body text and the published date for the article	Pass

O1T11	A user can't create an account for another user	The system should refuse to add the user	The system sent the following message: "unauthorize attempt to insert data"	Pass
O1T12	A user can update his own account	The user account must be updated	The user account updated	Pass
O1T13	A user can delete his own account	The user account must be deleted	The user account deleted	Pass

Oktob Level 2 Testing

Table 5-6: Level two testing results

Number of the test	**Tasks**	**Expected Result**	**Actual Result**	**Fail or pass**
O2T1	A user can add a comment on his articles	The comment must be added	The comment was added on the database.	Pass
O2T2	A user can add a comment on other user's articles	The comment must be added	The comment was added on the database.	Pass
O2T3	Non-register can't add comment	The system should refuse to add the comment	The system sent the following message: "unauthorize attempt to insert data"	Pass
O2T4	A user can delete his own comments	The comment should be deleted from the database	The comment was deleted from the database.	Pass
O2T5	A user can update his own comments	The comment should be updated	The comment was updated	Pass

O2T6	A user can read any comment	The system should send the comment	The system Sent the comment	Pass
O2T7	A user cannot update another user's comment	The system should refuse to add the comment	The system sent the following message: "unauthorize attempt to update data"	Pass
O2T8	A user cannot delete another user's comment	The system should refuse to delete the comment	The system sent the following message: "unauthorize attempt to delete data"	Pass

Oktob Level 3 Testing

Table 5-7: Level three testing results

Number of the test	**Tasks**	**Expected Result**	**Actual Result**	**Fail or pass**
O3T1	Register user can add an article to his reading list	The article should be added to his reading list	The article was added to his reading list	Pass
O3T2	A user can delete an article from his reading list	The article should be deleted from his reading list	The article was deleted from his reading list	Pass
O3T3	A user can read his reading list	The system should send all article in his reading list	The system sent all article in his reading list	Pass
O3T4	A user can't read other users reading list	The system should refuse to read a user reading list	The system refused to read a user reading list	Pass
O3T5	Non-register users can't add a reading list	The system should refuse to add a reading list	The system refused to add a reading list	Pass
O3T6	A moderator can add a user	Create a user account in the database	The account has been added to the database	Pass

O3T7	A moderator can't delete users	The system should refuse to delete a user account	The system sent the following message: "unauthorize attempt to delete data"	Pass
O3T8	A moderator can add a new article	The article must be added to the database	The article has been added to the database	Pass
O3T9	A moderator can add a comment	The comment must be added	The comment was added on the database.	Pass
O3T10	A moderator read all resources	The system should send any resource a moderator wants	The system sent all resources a moderator wants	Pass

Oktob Level 4 Testing

Table 5-8: Level four testing results

Number of the test	**Tasks**	**Expected Result**	**Actual Result**	**Fail or pass**
O4T1	Register user can add like	A like must be added	like was added to the database	Pass
O4T2	A user can delete his own like	like must be deleted	A like was added to the database	Pass
O4T3	A user can see all likes on the article	The system should send all likes on the article	The system Sent all likes on the article	Pass
O4T4	Author can publish his article	The article should be published	The article was published	Pass
O4T5	Author can unpublish his article	The article should be deleted from published list	The article was deleted from published list	Pass
O4T6	Premium author can add premium article	The premium article should be added	The premium article was added to database	Pass
O4T7	Author can add free article	The free article should be added	The free article was added to database	Pass
O4T8	Premium author can update his premium article	The article should be updated	The article was updated	Pass

O4T8	Register user can read the free article	The system should send the free article	The system sent the free article	Pass
O4T9	Non-register user can't read the free article	The system should refuse to read an article	The system sent the following message: "unauthorize attempt to read data"	Pass
O4T10	Non-register user can't read the premium article	The system should refuse to read an article	The system sent the following message: "unauthorize attempt to read data"	Pass
O4T11	The user who paid can read the premium article	The system should send the premium article	The system sent the premium article	Pass

CHAPTER 6

6. CONCLUSIONS AND FUTURE WORKS

6.1. Conclusion

Our proposed system enables the users to act as super admins over their own data, where the actual network provider has no control over the users' data, this will improve user privacy and security as the users are the only person who can grant access to any other user to access the data. The system is flexible, allowing fine-grain role-based access control to resources. Initial testing indicated the system is secured and also easy to use. For future works, the system UI needs more improvements and work to make it usable and ready to be used by normal users

The Etqan system can auto-generates API from a JSON schema that can represent an ER diagram, it also allows users to apply access-control to these resources, having the ability to easily to create an API with access control policies on any resource this can be integrated with social media which will give the user the full control of their own data as they can create any API easily and manage access to their own data the way they decide.

The system tests and evaluation results show that the system can be used in simple to large scale applications and can handle a high level of complexity in the program design and in the entities relationships.

Etqan system has proved its ability to create custom API that can be integrated with social media; however, the system still needs further works and development in order to be a full-featured system that can cope with the requirements of most users, future works to be carried out on the system include, add the ability to create API without any user authentication, and added a validation rule to match password or any other two values or Add different methods of authentications other than the basic auth, which include (JSON access token, cookies, o-auth) and add authentication for a device. , added validation rule for a value to be within a set of values, and added the ability to set project settings from JSON files such as languages, pagination size, etc.

Add a feature to allow users to change their role according to their activities and results in the system, for example, an author can become a senior author when it has at least 1500 experience points, and be allowed to check bandwidth and quotas and set usage limits for each user, and Integrate Etqan system with Facebook API, twitter, Instagram, eBay, Amazon store, Google Firebase, Amazon Cloud services.

6.2.How the system can be used with social networks?

A user creates an API to expose the data online for example an API to manage user images, The user can connect to each social network using that social network SDK, for example, Etqan will implement Facebook SDK, The user authenticates his app with Facebook using Etqan UI, then fetch the list of his friends on Facebook, Users can create a new user for each friend and assign them roles, When a friend comes to the API and authenticates with his Facebook profile, the generated API will authenticate the user and assign his role so the friend will be able to perform actions according to the role given.

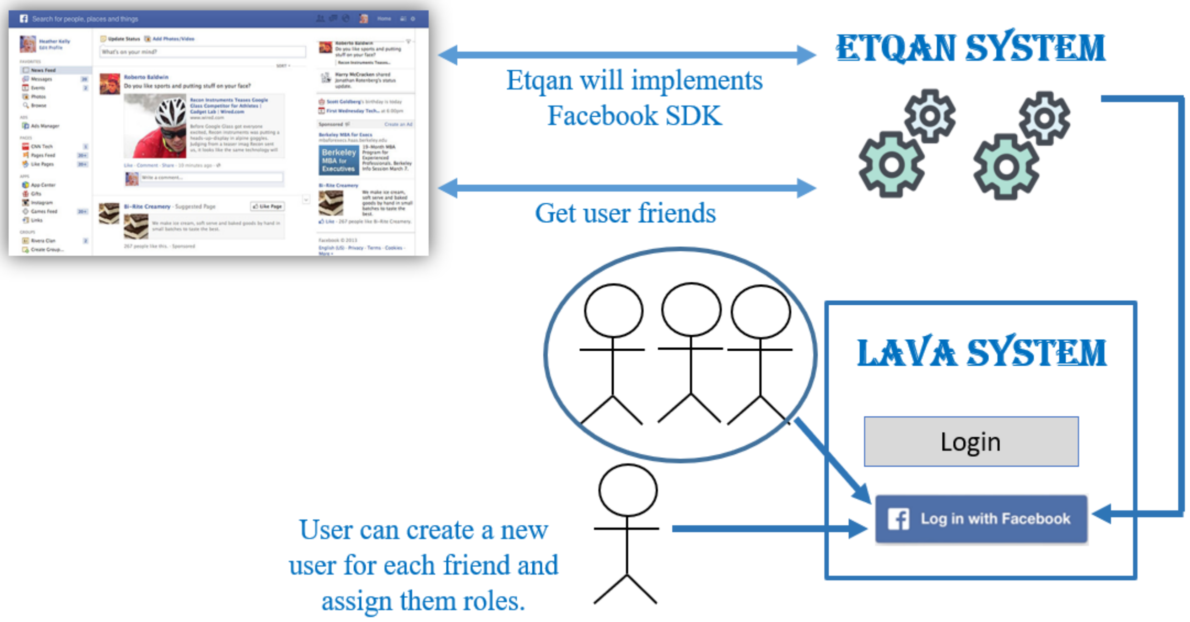

ETQAN SYSTEM
Etqan will implements Facebook SDK
Get user friends
LAVA SYSTEM
Login
Log in with Facebook
User can create a new user for each friend and assign them roles.

Figure 6.1: using Etqan in social networks

Appendix A

The Questions of the questionnaire, The Questionnaire entitled " Studying the extent of awareness that the use of social networks has about privacy and security on social networks, and the problems facing the user of these networks in terms of defining powers and privacy".

What is your goal in using social networks?	• 43.6% To share photos. • 64.1% Follow the news of friends. • 79.5% Follow the news of the country • 25.6% Buying and selling. • 38 % Professional communication with co-workers. • 38% To share his/her activities with others.
Do you share your personal information or private photos on social networks?	• 51.3% Yes. • 46.2% No. • 2.6% I do not know.
Do you know all of the people who can see your information on social networks are?	• 43.6% Yes. • 48.7% No. • 7.7% I do not know.
Have you ever posted a picture or information and reached people you don't know on social networks?	• 41% Yes. • 43.6% No. • 15.4% I do not know.
Do you feel fear for your information and mistrust when using social networks?	• 66.7% Yes. • 23.1% No. • 10.3% I do not know.
Do you think social networks keep your information private?	• Yes. • No. • I do not know.
Has the leak of your information on social networks ever caused moral or material harm?	• 74.4% Yes. • 12.8% No. • 12.8% I do not know.

If your personal information was leaked on social networks, the reason was:	• 61.5% No information was previously leaked to me. • 12.8% Not knowing the correct privacy settings. • 15.4% My personal information has been used by social media sites as a target for sponsored ads or promotions. • 10.3% I have not changed the default privacy settings.
Through your knowledge about social networks, the information that you put on the sites:	• 43.6% It remains the property of the user and no one can control it. • 56.4% It becomes the property of the site owners and they can dispose of it.
Have you heard anything about the privacy of social networks?	• 59% Yes. • 41% No. • 0% I do not know.
Do you use privacy settings on social networks to protect your data?	• 76.9% Yes. • 20.5% No. • 2.6% I do not know.
Before you post a photo or a sensitive post, do you look at the privacy settings?	• 59% Yes. • 41% No. • 0% I do not know.
Do you know how to determine the powers of people on social networks?	• 53.8% Yes. • 41% No. • 5.1% I do not know.
The most important reason for your need for privacy on social networks	• 15.4% Protection of social reputation (hide my information to protect my social reputation). • 76.9% Prevent identity theft and protect information from social network owners and third parties. • 7.7% Physical protection by protecting my position and my appearance.
Do you know how to change the privacy settings on social networks?	• 53.8% Yes. • 41% No.

	• 5.1% I do not know.
If you get a privacy error, have you changed your behavior? How do?	• 56.4% Yes I know how to change the settings. • 25.6% I don't know how to change the settings. • 17.9% Social networks don't have the privacy settings I want.
I would prefer to have a system independent of the owners of social networks that provides methods to protect my information on social networks and that I control it.	• 80% I agree. • 15% Neutral. • 5% Disagree.
I will use a system that gives me the freedom to control my information on social networks if it is available	• 78% I agree. • 17% Neutral. • 5% Disagree.

The link of Etqan project on the bitbucket website:

https://bitbucket.org/awad99/etqan-api-maker/src/master/

The link of Grant library on the GitHub website:

https://github.com/Talaween/Grant

The link of Oktob JSON file

https://drive.google.com/file/d/1-M1_6wPvJTOGwAbfzYFzbaJwlfkyGDu0/view?usp=sharing

7. REFERENCES

Ali, S., Islam, N., Rauf, A., Din, I. U., Guizani, M., & Rodrigues, J. J. P. C. (2018). Privacy and security issues in online social networks. *Future Internet*, *12*, 1–12. https://doi.org/10.3390/fi10120114

Awad, M., Al-Karaki, R., & Idais, D. (2021). User as a Super Admin: Giving the End-Users Full Control to Manage Access to Their Data in Social Media Networks. *2021 International Symposium on Electrical, Electronics and Information Engineering*, 623–627.

Barker, K., Askari, M., Banerjee, M., Ghazinour, K., MacKas, B., Majedi, M., Pun, S., & Williams, A. (2009). A data privacy taxonomy. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5588 LNCS,* 42–54. https://doi.org/10.1007/978-3-642-02843-4_7

Beato, F., Kohlweiss, M., & Wouters, K. (n.d.). *Enforcing Access Control in Social Network Sites*. 1–11.

Beye, M., Jeckmans, A. J. P., Erkin, Z., Hartel, P., Lagendijk, R. L., & Tang, Q. (2012). Privacy in online social networks. *Computational Social Networks: Security and Privacy, 9781447140,* 87–113. https://doi.org/10.1007/978-1-4471-4051-1_4

Boar, B. H. (1984). *Application prototyping: a requirements definition strategy for the 80s*. John Wiley & Sons, Inc.

Boyd, D. M., & Ellison, N. B. (2007). Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, *13*(1), 210–230. https://doi.org/10.1111/j.1083-6101.2007.00393.x

Cai, F., Zhu, N., He, J., Mu, P., Li, W., & Yu, Y. (2019). Survey of access control models and technologies for cloud computing. *Cluster Computing*, *22*(3), 6111–6122.

Cao, H., Falleri, J. R., & Blanc, X. (2017). Automated generation of REST API specification from plain HTML documentation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10601 LNCS*, 453–461. https://doi.org/10.1007/978-3-319-69035-3_32

Carminati, B., Ferrari, E., Heatherly, R., & Kantarcioglu, M. (2010). Semantic web-based social network access control. *Computers & Security*, *30*(2–3), 108–115. https://doi.org/10.1016/j.cose.2010.08.003

Cheng, Y., Park, J., & Sandhu, R. (2016). An Access Control Model for Online Social Networks Using User-to-User Relationships. *IEEE Transactions on Dependable and Secure Computing, 13*(4), 424–436. https://doi.org/10.1109/TDSC.2015.2406705

Ed-douibi, H., Luis, J., Izquierdo, C., Gómez, A., Tisi, M., & Cabot, J. (n.d.). *EMF-REST : Generation of RESTful APIs from Models*.

Fielding, R. T., Taylor, R. N., Erenkrantz, J. R., Gorlick, M. M., Whitehead, J., Khare, R., & Oreizy, P. (2017). Reflections on the REST architectural style and" principled design of the modern web architecture"(impact paper award). *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 4–14.

Fuchs, L., Pernul, G., & Sandhu, R. (2011). Roles in information security - A survey and classification of the research area. *Computers and Security*, *30*(8), 748–769. https://doi.org/10.1016/j.cose.2011.08.002

Galić, I. (n.d.). *Performance Overhead of Haxe Programming Language for Cross-Platform Game Development*. *6*(1), 9–13.

Gurses, S., Rizk, R., & Gunther, O. (2008). *Privacy design in online social networks: Learning from privacy breaches and community feedback*.

Imran-Daud, M., Sánchez, D., & Viejo, A. (2016). Privacy-driven access control in social networks by means of automatic semantic annotation. *Computer Communications, 76*, 12–25.

Kühnhauser, W. E., & Pölck, A. (2011). Towards access control model engineering. *International Conference on Information Systems Security*, 379–382.

Li, E. (1990). Software testing in a system development process: A life cycle perspective. *Journal of Systems Management*, *41*(8), 23.

Lynn, T., Mooney, J. G., Werff, L. Van Der, & Fox, G. (2021). *Data Privacy and Trust in Cloud Computing*.

Malley, J., Philippopoulos, A., & Woitek, U. (2009). To react or not? Technology shocks, fiscal policy and welfare in the EU-3. *European Economic Review*, *53*(6), 689–714.

Mansour, E., Sambra, A. V., Hawke, S., Zereba, M., Capadisli, S., Ghanem, A., Aboulnaga, A., & Berners-Lee, T. (2016). *A Demonstration of the Solid Platform for Social Web Applications*. 223–226. https://doi.org/10.1145/2872518.2890529

Osborn, S., Sandhu, R., & Munawer, Q. (2000). Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, *3*(2), 85–106.

Pang, J., & Zhang, Y. (2014). A New Access Control Scheme for Facebook-Style Social Networks. *2014 Ninth International Conference on Availability, Reliability and Security*, 1–10. https://doi.org/10.1109/ARES.2014.9

Praveena, A. K. R., Bhavani, B. D. S. D., & Babu, C. K. S. (n.d.). Online Social Networks for a Multiparty Access Control Model and Mechanisms. *International Journal of Computer Science & Network Solutions*.

Queirós, R., La, I., & Porto, D. I. E. P. (2020). *Kaang : A RESTful API Generator for the Modern Web*. *1*, 1–15.

Sachan, A., & Emmanuel, S. (2011). Efficient Access Control in Multimedia Social Networks. In *Social Media Modeling and Computing* (pp. 145–165). Springer.

Sohan, S. M., Anslow, C., & Maurer, F. (2015). *SpyREST : Automated RESTful API Documentation using an HTTP Proxy Server*. 271–276. https://doi.org/10.1109/ASE.2015.52

Tapiador, A., Carrera, D., & Salvachúa, J. (n.d.). *Tie-RBAC : An application of RBAC to Social Networks*.

Vasenin, V., Itkes, A., Krivchikov, M., & Yavtushenko, E. (2020). ChRelBAC data access control model for large-scale interactive informational-analytical systems. *Journal of Computer Virology and Hacking Techniques*, *16*(4), 313–331.

Vecchiato, D. A., De Toledo, M. B. F., Fantinato, M., & De Souza Gimenes, I. M. (2010). Electronic contract negotiation and renegotiation using features. *WEBIST 2010 - Proceedings of the 6th International Conference on Web Information Systems and*

Technology, *2*(Webist), 313–318. https://doi.org/10.5220/0002800203130318

Wang, S., Keivanloo, I., & Zou, Y. (2014). How do developers react to restful api evolution? *International Conference on Service-Oriented Computing*, 245–259.

Wang, Y. (2021). Fine-grained access control model based on sensitivity calculation by fuzzy mathematics. *Journal of Physics: Conference Series*, *1856*(1), 12024.

Wazza, J. J., Bruns, A., West, W., Weaver, A. C., & Morrison, B. B. (n.d.). *Social networking Networking*.

Zheleva, E., & Getoor, L. (n.d.). *Chapter 10 PRIVACY IN SOCIAL NETWORKS : A SURVEY*. https://doi.org/10.1007/978-1-4419-8462-3