NAME	:	**NGUYEN HOANG MINH**
UOG ID	:	**001089370**
COURSE	:	**COMP1787 – REQUIREMENTS MANAGEMENT**
COURSEWORK TITTLE	:	**AN INVESTIGATION OF GRAPHQL INTO DEVELOPING SOCIAL NETWORK**
INSTRUCTOR	:	**LAM NGUYEN TRUNG NAM**
DUE DATE	:	**MAY 13TH 2020**
CENTER	:	**FGW HCM**

Date: 11/05/2020

Acknowledgement

Sincerely thank to my family and my parents for creating the motivation and conditions for me to complete this thesis. I also would like to express my gratitude to the teachers and FPT University for creating the best conditions and helping me in times of need, having dedicated to teaching and transmitting knowledge and experience to my academic and mature.

Although there are many efforts in the learning process, as well as in the process of making the graduation project, it is inevitable shortcomings. I am looking forward to the valuable suggestions of all reviewers; you let my results be more complete.

Once again, I sincerely thank!

Nguyen Hoang Minh

Table of Contents

Source code

GitHub link for source:

https://github.com/nguyenhoangminh2401/final_year_project.git

Database connection:

Host: fyp-mysql.cmopyyx4mmg8.ap-southeast-1.rds.amazonaws.com

Port: 3306

Database: mydb

Username: admin

Password: 12345678

Chapter I: Introduction

1. Project objectives

In the current social context, with the continuous development of information technology, the Internet is gradually asserting the importance, in which appear today. A number of social networking sites offer individuals and organizations the opportunity to share their information, but it is also a challenge for specialized management agencies to ensure content and scope of activities. In most of us, social networks like Facebook, Zalo, YouTube have quickly becoming an important part of many people, especially for the younger generations, social networks have an even more important role and great influence on people. In particular, social networks (SN) have been and will be a part of social life in a public section. Nowadays, when information technology develops, no one can deny the benefits that social networks bring, especially young people.

GraphQL is a query language created by Facebook with the purpose of building applications for customers based on intuitive and flexible syntax to describe requests and interact data.

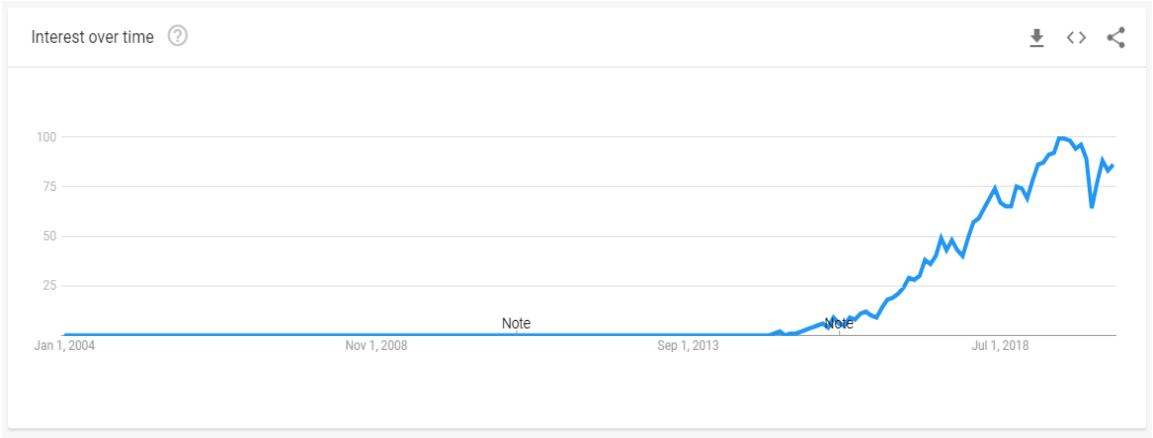
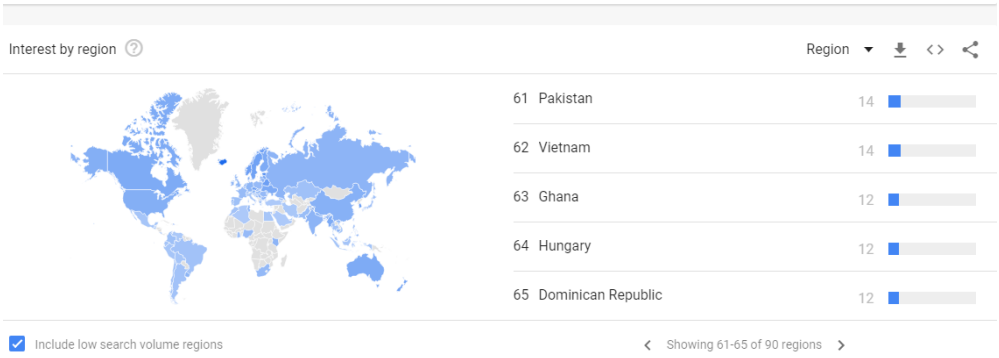


Figure 1. Graphql trends (as of May 5, 2020) Source: Google trends

Graph of Google Trends shows that Graphql has been steadily increasing over the past years (Figure 1), proving that Graphql is developing well. Although Graphql is not as 'hot' as React /

Vue / Angular with a large number of users, that does not mean that Graphql receives less attention from user/ developer. Search trend results also show that Vietnam ranks 62nd among the countries searching for GraphQL keywords.



If we wonder which an area that makes people dizzy, it must be ‘knowledge update’. Technology is constantly changing; programmers must constantly learn new knowledge to catch up with the times. Therefore, the goal of this research is to discover about GRAPHQL, and simulate how the applicability of GraphQL applies to social networks.

2. Project plan

Software project management is the activities in planning, monitoring and control of project resources (e.g. cost, people), execution time, project risks and project implementation process judgment; to ensure the success of the project. Software project management needs to ensure weighing equal to three factors: time, resources and quality.

Problems often occur with a software project:

- Project implementation time exceeds expected level.
- Project implementation cost exceeded plan.
- The project results are not as expected.

- The responsibility of the project manager:
- Time management: Schedule, check and compare project implementation with schedule
- Resource management: identify, allocate and coordinate resources
- Product management: add and remove functions in accordance with customer requirements
- Risk management: identifying, analyzing risks and proposing solutions
- Organize how to work.

PREMILINARY PROJECT PLANING

While the project has not been completed or has been canceled, the implementation is repeated the following work:

- Schedule project implementation.

- Perform scheduled activities.
- Track the progress of the project, compare it with the schedule.
- Re-evaluate the project parameters.
- Reschedule the project implementation for new parameters.
- Re-negotiate binding and delivery products of each timeline.
- If problems arise, review the initiation techniques that provide the necessary measures.

3. Project outcomes

3.1.General

Need to demonstrate excellent problem-solving analytical skills and technical aptitude, especially in the field of software development. In this way, we will gain an in-depth knowledge of the application of knowledge into the real work environment. Through this project, the project helps students gain confidence when approaching new knowledge and ability to solve problems raised in the field of information technology. Ability to self-plan, build solutions and deploy an enterprise project at the request of customers; Analyzing, designing and implementing management information systems for small and medium-sized enterprises; Evaluate the quality and performance of information systems; optimize and organize maintenance plans; Implementation of system integration, improvement and technology transfer; Advice on security, technical and technological solutions; Consulting on information system design, software and network development.

3.2.Details outcome

- Research/ discover how can GraphQL be implemented into API.
- Outline / list functional requirements for a social networking application (basic requirements).
- Research which database is best suited for the project.

- Discover front-end technologies that lead front-end development technologies (Ex. Vue.js, Bootstrap)

3.3.Project evaluation

Contribute to the theory and practice of the thesis:

+ *Theory*: The study aims to contribute to prove the usefulness of GraphQL.

+ *Practice*: Offer reviews, comments and solutions to help improve and improve the efficiency for building social networking applications.

Chapter II: Literature Review

1. Introduction

The advent of web generation 2.0 makes a real revolution in the internet world. It is a revolution not only in technology but also in the way in which everyone participates in contributing to virtual society to create a community environment, not merely “browsing and viewing” as before. In particular, social networking has gradually become an integral part of modern life. It gives people the opportunity to connect easily, to share interests, habits and thoughts. Social networks are increasingly developing widely and proving their charisma and role in all aspects of social life such as commerce, study and entertainment.

Social networking, also known as virtual social network, the service connecting members of the same interests on the Internet together for many different purposes irrespective of space and time. According to Boyd and Ellison (2007), social networks are “web-based services that allow individuals to build a public or public profile”. When a new social network was born, we seemed to focus on two main issues: a social network of Vietnamese people working for Vietnamese people and how to compete with Facebook. It is easy to see that search engines like Google, Facebook social networks from abroad have attracted most users in Vietnam. So why do we still have to think of building domestic tools that compete with platforms that already have so many users?

First of all, value information with users. We can see that foreign companies can provide or accept up-to-date, accurate information. Even a leading company in the search of map data is Google, the location data, aerial photos from many regions in Vietnam have not been updated. This natively systems can do better. Google itself has to spend money to buy or collaborate to get this

information from a number of companies in different countries. Matching, ranking, and filtering display content on platforms like Facebook in some areas can still be improved, but they may know it without doing it because it affects their economic benefits and the majority of their other users. The minority they overlook is probably a Vietnamese user, a Vietnamese business. Building a domestic platform with accurate, rich and essential information content, a good way of displaying information can still appeal to users compared to platforms from other countries.

Second in terms of security. A search system or social network can contain information to help prevent criminal cases as Internet users increasingly depend on it as part of their lives. If Google and Facebook do not cooperate to help countries solve this problem, then the countries must have their own tools to solve it. For example, an individual uses a search engine or social network to serve a bad thing (finding action objects, finding locations of action, finding ways to act or reporting frauds, false charges If these platforms do not provide data to the Vietnamese security agencies, it will be difficult for the authorities to solve criminal cases, causing harm to society and national security.

Third, economic benefits, nothing is free. Big tech companies like Facebook have a good tech background, so when there's a lot of data they can guide their goals. Our users post information, click (click), view (views) so that they use those interactions to benefit a lot from advertising and even have a news service for third parties to make a profit. If there is a lack of competition then people have to accept to pay high prices for foreign businesses. We once knew a Vietnamese business would be willing to give another $ 50,000 to another Vietnamese business if it helped them get the number of ad views just close to the same number Facebook gave because they clearly saw too much money for foreign companies, but may not be able to target the right audience.

- **The characteristics of social networks**

- *User*

The online social network is built and driven by the users themselves. User will determine the content of online social networking sites. The orientation of that content is determined by anyone participating in the discussion. This is what makes the fun and dynamism social networks bring to internet users

- *Personal*

On social networking sites, each member has a profile with a profile of their own.

- *Interaction*

Another feature of modern social networks is interaction. Users on social networking sites can communicate easily and participate in online app together.

- *Rely on the community*

Social networks are built and maintained based on the characteristics of the community; groups are established based on interests and beliefs.

- *Relationship development.*

The community on online social networking sites is an open community, where users are free to choose and develop their relationships. The more relationships you have in the network, the more relationships it establishes based on existing relationships.

- *Emotionally superior content*

A unique feature of social networks is the emotional element. While the previous site has focused primarily on providing information to visitors, social networks really provide users with a sense of security to share information and a sense that no matter what is too difficult. , deadlocked, their friends are always with them, listen to them anytime.

2. GraphQL – The heart of this project.

According to the official definition of the GraphQL developers, it is a query language for API and also a server-side runtime for making execution of queries through a system of type that help us define the probably data.

However, GraphQL could be understand as a new generation of data transfer architectural style if compared to REST (REpresentational State Transfer), architectural style was first presented by Roy Fielding in 2000 and has been widely chosen in developing web service for nearly twenty years. For explanation, as its developers mentioned, GraphQL is not tied to any specific database system or storage engine, instead, it is backed by our existed code and data models.

2.1.A brief summary of GraphQL development:

GraphQL is an open-source data query and manipulation language for APIs, as well as a fulfilling queries runtime with existed data, was developed internally by Facebook since 2012. Later on, in 2015, GraphQL was released in public and being collaborating developed by other big tech company until today.

On November 7th of 2018, GraphQL project was officially moved to new established GraphQL Foundation, which is hosted by the non-profit foundation of Linux. For the first time of six years since it followed the adoption timelines that set out by the creator, Lee Byron, GraphQL is coming close to its goal, which is became omnipresent across web platforms.

GraphQL provides an approach to developing web server data APIs, and has often been compared and contrasted with REST or other web service architectural styles. The reason for this comparison is it allow clients application to define the structure model of required data, and the exactly the same structure of data would be return from server instead of the traditional “make a call” proceed.

On 9 February 2018, the GraphQL Schema Definition Language (SDL) start to became a part of GraphQL specification.

2.2.Why use GraphQL?

- **As for what we need, get exactly that**

GraphQL provide us the capability of sending a single query to back-end data API and get exactly what we have described, nothing more and nothing less. Therefore, GraphQL always return predictable results. As a result, application using GraphQL are fast and stable because they completely control the data structure they need to get, not the server.

Request

```
{
  person {
    name
    height
    age
  }
}
```

Response

```
{
  "person": {
    "name": "Nguyen Hoang Minh",
    "height": 1.75,
    "age": 24
  }
}
```

- **Get variety of resources in one single request**

GraphQL queries could access not only the properties of one specified resource but also smoothly follow the complicated references between their models. Opposed to typical REST APIs that require calling data from multiple of URLs route, GraphQL data APIs could get all the data our application needs in a single HTTP request. Thus, applications that using GraphQL could operate smoothly even in slow mobile network connecting conditions.

- **Describe what is possible with a fulfill type system**

GraphQL data APIs are organized in the terms of datatypes and fields, not endpoints. Thus, GraphQL has the ability to access all the full capabilities of our data through a single endpoint. In

order to do that, GraphQL uses type system to ensure that application only ask for what is possible as well as provide probably and useful errors. Moreover, application could use type system to avoid writing manual parsing code problem.

Client

```
{

   user {
     fullname
      followers {
       name
       country {
        name
        location
       }

      post {
        title
        content
       country {
        name
       }
      }
     }
    }
```

Server

```
type Query {
  user: Profile
}

type Profile {
  fullname: String
  followers: [Profile]
  country: Country
  post: [Post]
}

type Country {
  name: String
  location: String
}

type Post {
  title: String
  content : String
  country: Country
}
```

- **Moving faster with the efficiency provided developer tools**

GraphQL is providing us a way to detect exactly which data that could be requested from our API right in the code editor, identify the possible issues or syntax errors before making request of queries, and taking the advantage of developed code intelligence system; by building powerful tools using the leveraging result of our data API types.

- **Evolve the data API without versions**

GraphQL allows us to add new fields and types to our GraphQL data API with out making any impact to the existed queries. Besides, the aging field could be disabled or hidden. Developing in

one single version give application the ability of continuously append new features and help server code become clear and maintainable.

Evolve 1

```
type Post {
  title: String
  content: String
  createAt: DateTime
}
```

Evolve 2

```
type Post {
  title: String
  content: String
  createAt: DateTime
+ hashtag: String
}
```

Evolve 3

```
type Post {
  title: String
  content: String
  createAt: String
  hashtag: String
+ author: String
}
```

Evolve 4

```
type Post {
  title: String
  content: Int
  createAt: String
  hashtag: String
  author: String
+ authoredBy: User
}

+ type User {
+   fullname: String
+   post: [Post]
+   followingPost: [Post]
+ }
```

Evolve 5

```
type Post {
  title: String
  episode: Int
  createAt: String
  hashtag: String
- author: String @deprecated
  authoredBy: User
}

type User {
  name: String
  post: [Post]
  followingPost: [Post]
}
```

- **Migrate our code and data**

As mentioned above, GraphQL is not tied to any database system or storage engine. So, it creates a single-unique-formed data API across all of our application without any limited.

Because the available of GraphQL engines in many languages, we can leverage the existed data and code to provide functions for every field of type system and GraphQL engines could use them in optimized concurrence.

```
type User {
  fullname: String
  country: Country
  followers: [User]
}
```

Java

```
// type User {
class User {

  // fullname: String
  getFullName() {
    return this._fullname
  }

  // country: Country
  getCountry() {
    return
fetchCountry(this._countryID)
  }

  // followers: [User]
  getFollowers() {
    return
this._followersIDs.map(fetchUser)
  }
}
```

C#

```
// type User {
public class User {

  // fullname: String
  public String Fullname { get; }

  // country: Country
  public async Task<Country>

GetCountryAsync() {
    return await GetCountryAsync
(_countryID);
  }
  // friends: [Character]
  public async
IEnumerable<Task<Character>>
GetFollowersAsync() {
    return _
_followersIDs.Select(FetchUserAsync);
  }
}
```

- **Who are using GraphQL?**

We cannot say about GraphQL without mention its “hometown”, Facebook, the first and the biggest user of GraphQL engines. Next, we have GitHub - The world's leading software development platform, as they said about themselves, along with the photo sharing social network like Instagram or Pinterest are all integrating GraphQL into their data system with other nearly a hundred of adopter company.

2.3.Within a competitive with REST!

REST has been widely used by lots of developers and became a traditional way to transfer data through HTTP protocol. On the other hand, as mention, GraphQL is the next generation of data transfer architecture that generally regarded a replace for REST.

So, the question is, how do both API architectural styles compete against each other?

Let us dig in to figure out the similarities and differences between them: the good, the bad and the ugly.

- **Firstly, a brief introduction about REST:**

Generally, REST is an API architecture, used to develop web application services that allow system to access and manipulate the textual representations of web resources through a pre-identified collection of stateless operations:

GET **POST** **PUT** **DELETE**

In REST architecture, the development of client-side and server-side usually occur independently. Which mean, the front-end code could change without causing any impact to the server-side operation. Thus, the modular and separate of overall system are preserved.

The basic idea of REST is allocating the resource into defined URLs. For example, through a GET request as following to URL "*GET /post/1*" and get a responded JSON string.

GET /post/1

```
{
          "title": "Hello",
           "content": "My name is Ming Hwang Weng",
          "createAt": "January 24, 2020",
          "author": {
                 "firstName": " Ming",
                 "lastName": " Weng",
                 "country": "Vietnam",
                 "education": "University of Greenwich"
          }
  }
```

As we can easily see, the type of resource and the way to transfer that exactly resource are coupled.

- **The similarities and differences between REST and GraphQL:**

First of all, they are both developed based on the idea of creating a resource then specify an ID into that resource. Second, GraphQL and REST are also fetching data through HTTP request and return JSON type data.

However, the URLs, also known as endpoints, is the data object identity in REST, while they have nothing to do with the data structure in GraphQL. In other words, in REST, the object is defined on the server-side, while it is defined on the client-side in GraphQL. Moreover, GraphQL is single endpoint, so the requests for any resource would always point into a single URL. For example, in REST, we might meet something like:

```
GET api.example.com/users/1?include=followers.follower.name&friend.country=Vietnam
```

While GraphQL provide us a cleaner way to present it:

```
{
 user(id: 1) {
   followers {
     followers(country: "Vietnam") {
       name
     }
   }
 }
}
```

So, that should be a slightly winning for GraphQL in this competition. However, let us start to compare two of these architectures.

- **Round 1: Fetching**

Nowadays, web and mobile application required larger and larger sets of data that generally are the combination of multiple related resources. But the problem of the traditional REST-based data API is, to fetching all the required data from multiple resource, we might have to access N different endpoints that causing a multiple trip N+1 rounds, or other words, a chain of process, before all the data are represented to a formed displayable object. For example, if we need to get the data from two or more different endpoints, we have to make that much requests to our API server, otherwise creating a new endpoint to serve only this requirement. Both of that solution always lead to one of these problems, slowing down of performance or development speed. On the other hand, GraphQL can solve this issue by enable the server to aggregate the set of response data that the client request for in a single query. By using a strong type system that tell both client and server

what are the structure of data and how to access it, each client-side application can specifies the probably what they need, gain the insight into data request, define what fields should be available to use and what fields would be deprecated in order to reduce the redundancy.

Another common limitation that we face when implement the REST architecture is the **over and under fetching**. The reason for this problem is client only could get the data by calling the endpoints that return fixed data objects, but the UI is flexible and we could not create specified endpoint for every of them. As a result, client application is not able to get exactly what they and lead to the situation of the **over and under fetching.**

To understand more two mentioned problems, we should have an example. Let say we are creating a social network, and we are going to get all the latest posts.

GET /api/posts

```
[
{
  "title": "This is the first title",
  "content": "Hello, I am the content of the first post.”
  "createAt": "$178"
},
{
  "title": " This is the second title ",
  "content ": "Hi, I might be the content of the second post.",
  "createAt ": "$299"
}
]
```

But what if we need to display the posts ‘s authors information. There would be three option to consider:

Option 1: Fetching posts data from another different endpoint.

GET /api/ posts /:id

```
{
   "posts": {
     {...}
     "author": {
       "name": "Minh Nguyen",
       "email": "minhnhgcs16319@fpt.edu.vn"
     }
   }
 }
```

Problem of REST: If choosing this option, we are running into an under-fetching situation. In order to display the author information, we have to run another server request for each item or users, in others words, if we have N number of posts, we have to make N+1 request to form the final displayable result. This would cause a serious performance issue later on, when your data scale up.

Option 2: Modify the existed endpoint to return plus the authors information

GET /api/posts

```
[
    {
      {...}
      "author": {
        "name": " Minh Nguyen",
        "email": "minhnhgcs16319@fpt.edu.vn"
      }
    },
    {
      {...}
      " author": {
        " name": "Nam Nguyen",
        " email": "nguyenvannam@gmail.com"
      }
    }
  ]
```

Problem of REST: Well, this might be a perfect solution… at first! But if we need to reuse this endpoint to display posts somewhere else without their authors, such as the Instagram exploder, for example, then it is leading to a mentioned problem of over-fetching.

Option 3: Create new endpoint that return the posts details include their author information.

GET /api/postsWithAuthors

```
[
    {
      {...}
      "author": {
        "name": " Minh Nguyen",
        "email": "minhnhgcs16319@fpt.edu.vn"
      }
    },
    {
      {...}
      " author": {
        " name": "Nam Nguyen",
        " email": "nguyenvannam@gmail.com"
      }
    }
  ]
```

Problem of REST: The major problem of this option which is we have to create a new endpoint for each requirement such as UI or data process that needed to be done at the client-side. By this way, as mentioned, the development speed would be decrease proportional to the complication of front-end requirement.

Option 4: USING GRAPHQL

GET /api

```
query
{
       posts {
         title
         content
         createAt
         author {
           name
           email
         }
       }
     }
```

As it is easily seen, with GraphQL, front-end applications are able to define the structure of data and fetch exactly what they required with making any adding or modifying to server within one single query. Additionally, if we need some light weight data set as mentioned in **Option 2**, just create a new query like following with exactly the same endpoint as previous:

```
query
{
       posts {
         title
         content
         createAt
       }
     }
```

In conclusion, in this first round, GraphQL might be a little ahead thanks to the query and data type that help solving the inherent problems of REST, which are: multiple rounds calling and under-over fetching.

- **Round 2: Performance**

In common, REST architecture work in focus on keeping a service reliable to its major objectives, on the other hand, while the performance takes highest priority in GraphQL. As we have

investigated above, REST usually have to choose to face a problem between taking more than need or making multiple call that both lead to the performance problems.

In contrast, GraphQL work in aim to the smallest possible data return, which mean NO WASTED BITS are transferred over the protocol. In details, GraphQL simplifies the request by making back-end services to combine all the data according to described structure of the query, which make a result that fewer redundancy data being sending over the wired.

In conclusion, this should be considered as another winning round for the GraphQL. GraphQL is faster, that is the thing REST have to admit. The less data transfer and no multiple duplicated request always creates better performance.

- **Round 3: Caching**

Caching is a built-in mechanism of HTTP protocol that REST architecture was aimed to leverage. In details, the semantics of REST such as POST or GET are created to enable the browser caches, intermediate proxies, and server frameworks. Caching is important at the level of network, which can help reduce the number of data traffic transactions to the server and keep the frequently accessed resource close to the client through CDN.

However, GraphQL, instead of follow the specified caching of HTTP, use the single endpoint mechanism. As a result, developers need to make sure the caching is implemented probably for such the non-mutable queries that required to be cached. Therefore, we need to choose the correct key for caching that sometimes might contain parts of the body contents. The tools such as Relay or Dataloader, nowadays, could understands GraphQL query better, but still not cover everything as REST for example mobile caching.

Concludes, the strong single endpoint of GraphQL is become a limitation in caching data, which help REST totally beat it in this round gain back the place in this competition.

- **Round 4: Security**

Once again, the advantage of GraphQL become its weakness. The major strength of GraphQL is the ability of allowing client-side application query whatever data they needed, this might become a serious problem when the API is open and we cannot control the third-party client query. In some situation, the complicated join queries could cause the sever slow down performance or maybe DDOS. On the other hand, REST architecture is constrained to match data model and indexing used that can avoid those situations. Error handling in GraphQL also more complicated when the typical default return is OK status code.

Well, seem we are getting back to a fair fight, let us get to the final factor and the most important factor that need to consider when choosing an architecture for our service.

- **Round 5: Development**

The development process of an application or service often separated into two major part: Implementation and Maintenance. As we have concluded, the GraphQL seem like help the services easier to maintenance thanks to the data type system that manipulate the return data with making any change to the server-side. However, in order to build such a flexible system in GraphQL, we need to define complexity with types, queries and resolvers, which make GraphQL not a suitable solution for simple and short-term development services. In contrast, those things would be done much separately and easier in REST.

- **Conclusion:**

GraphQL and REST are just simply 2 architecture to transfer data via HTTP. Although GraphQL probably has lots of advantages over REST, it is not the best choice in every situation. However, in developing social, the complication of UI and display data require an architecture that provide not only flexible return but also easily to maintenance that reduce to minimum the redundancy

data and able to deal with slow connection network. So, that why we choose GraphQL for this social network project.

3. Database for social network – MySQL or MongoDB?

In not only social network but also any software product development, picking the appropriate database or storage system is extremely important. Choosing an unsuitable set up could cost us lots of wasting time and money and put our numerous users in upset about the services. MongoDB and MySQL are two different types of database, which are NoSQL and SQL database, respectively. Both of them are excellent database when being used in their correctly expected ways, however, which one should be considered as a better option for building a social network.

3.1 What is MongoDB?

MongoDB is a NoSQL database, so, what is NoSQL?

NoSQL database is the database system where all related data stored in a single document. This is good database that help increase the retrieval speed but can cause inconsistencies when data are duplicated in multiple documents. For example, this is a document stored in MongoDB, or other words, a record in collection:

```
var myDocument = {
  _id: ObjectId("59803df3ff983094948bd291"),
  name: { first: "Thomas", last: "Cage" },
  birthday: new Date('July 24, 1911'),
  email: "thomas.cage@gmail.com",
  article: [ "Day of Sun", "Good Morning", "Worlds" ],
  numberOfLike : NumberLong(241000)
}
```

The above document has fields that following data types:

- **_id**: contain the `ObjectId`
- **name:** contain the *embedded document* that include the **first** and **last**
- **birthday:** contain the value of `Date` data type
- **article:** contain an array of strings.
- **numberOfLike:** contain the value of `NumberLong` data type

MongoDB database the storage rapidly growth in horizon by scalable data model and keep everything in a correct order. This type of database would extremely useful when we need to store a huge amount of data but required quickly successful responses.

However, the problem of MongoDB is not about storing, it is about reading and updating. When the number of documents is increased and the requirement for data become more and more complicated, MongoDB reveals it weakness in query and multiple data extraction, which are the well-known advantages of SQL database systems such as MySQL.

3.2.What is MySQL?

MySQL is a relational database that created based on the SQL (Schema Query Language), where, data is stored in normalized tables to avoid the duplication. SQL database is the perfect model to keep data consistent and reliable at all times such as our users ‘s data and their “complicated relation in social network”. Data in MySQL or any other relational data are often stored in tables that created by SQL language with data labels and types described as following example:

```
mysql> DESCRIBE user;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
```

```
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| email   | varchar(20) | YES  |     | NULL    |       |
| address | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| follows | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

Different to MongoDB or other NoSQL databases, horizontal scalable in MySQL is not a strength. Relational databases are preferred to adhere the **ACID** model that focus on maintain the data reliable over the development of services.

A	**C**	**I**	**D**
Atomicity	Consistency	Isolation	Durability

3.3.What is data in social network

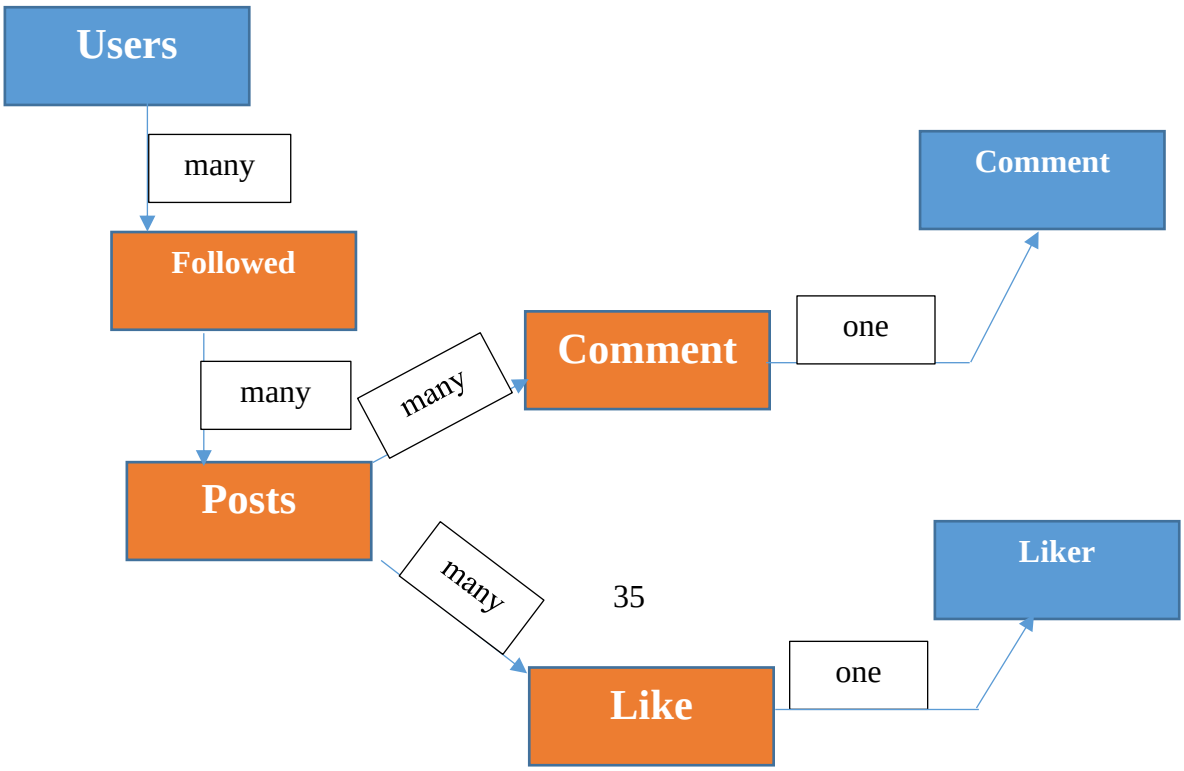
In order to make a clear point of view about social network, let talk about Facebook. It is simply a web application which is running on some sort of a server that allow us to connect to other people all over the world. There are some features of Facebook that we can easily see like a news feed that showing all your following people posts and many of random function placed both sides of this that we might never look at. As we already known from Facebook, social data are the information about every single user personal network that include their friends ‘s network, and

friends of their friends ‘s network and so on. Concludes, social network data is all about the complicated connection between users and their activities that directly or indirectly affect to each other, called activity stream.

3.4.How MongoDB stores social network data?

It is been a while since some people misunderstanding that social network data are not relational and NoSQL database like MongoDB is more suitable to use as a social network storage engine than other relational database. But, does it the truth?

As we have learned above, MongoDB is a document-oriented database, which meant instead of storing data in tables with individual records (rows), it stores them in the **collections** with individual **documents**. In MongoDB, every document often a big and complicated JSON string with out any specific format or structure of data. In a social network service, the one and only most important element is the user activity stream. The activity stream, like in Facebook is the list of all the posts of your following or related people that order by some particular factors such as: interest, interaction and most priority is the recent. Each post of activity stream is a nest of information such as likes, comments, reactions, etc. that usually lead us to another user.



Users has their friends, their friends have their posts, posts have like and comment and each of like and comment authored by a user, it is the network.

3.5.SQL database in developing social network?

Given that social data has many different relationships, especially with users, it lends better to relational databases over time. Although a non-query solution like MongoDB may seem like a great way to retrieve a lot of data quickly, the nature of user relationships in social networks can cause a lot of duplication.

Such copying allows data to become inconsistent and unreliable over time or queries become more difficult to handle if copying is deleted (because documents may need to be pointed. to other documents, not optimal for non-query database type).

Therefore, MySQL would be a better recommendation, since it would have the data reliability and relational tools needed to handle interactions and relationships between multiple users. You can also decide to use both MySQL and MongoDB together to use the best features of each database.

4. NodeJS and Express framework – Engine for the heart.

4.1.Introduction to NodeJS:

NodeJS (or Node.js) is well-known as an asynchronous event-driven JavaScript runtime, which is first designed to create the base server for scalable network services or application, like our social network project. This following code is a “Hello World” example of NodeJS.

```
const http = require('http');

const host = '192.168.0.1';
```

```
const port = 4000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World!');
});

server.listen(port, host, () => {
  console.log(`Server is now running at http://${hostname}:${port}/`);
});
```

As we can see, with NodeJS, variety of connections could be concurrently handled and each of those connection fire the callback if there is any work need to be one, otherwise, NodeJS stay asleep.

This concept of NodeJS is in a contradistinction to other modern well-known concurrency connection model, in which the operating system threads are chosen to handle the process. Thread-based networking is not comparative efficient and really to implement. Moreover, when using NodeJS, we are no longer worry about the processing dead-locking problems, because of a simply reason, there are no locks in NodeJS. To be more clearly, most functions of NodeJS do not perform I/O stream, thus, that why process should never block them. As there are no blocking, NodeJS is perfect for developing scalable and frequently update networking system services like social network.

NodeJS is influenced by systems like Ruby's Event Machine or Python's Twisted, so the designed concept is similar at some points. However, the event model is taken to a little further in NodeJS, where a loop event is presented as a runtime construct instead of as a library. For example, in many

other runtime systems, there should be usually required a blocking call in order to begin the loop event.

Generally, at the begin of scripts, callbacks are created to define behavior, then, the server is started by triggering a blocking call at the end of scripts with something like:

```
ServerEventEmmiter=>:run()
```

Contrast, in NodeJS, we do not need to start any loop call. NodeJS engine just simply enters the loop event right after finish the execution of input scripts and exits the loop event whenever no longer any callbacks to execute. So, it is similar to the behavior of browser JavaScript that hide the loop event form user.

HTTP, which is designed to aim for streaming and low latency, is the best option to integrate when developing in NodeJS. The combination helps NodeJS become suitable for building the core foundation of web application library or framework. Some of the well-known web library or framework that based by NodeJS such as: AdonisJS, Meteor.js, Nest.js, etc. And the most famous of them all: Express.js with over 41 thousand stars on GitHub.

As mention, NodeJS is developed without using the OS threads to handle the major process. Anyway, it does not mean we cannot leverage the advantage of multiple cores processing system in our environment. Child processes can be spawned by using the built in API like:

```
child_process.fork()
```

These API are built based on the same interface is the cluster module and designed to be easy to communicate with. They allow us to enable the load balancing mechanism over our environment system cores by sharing sockets between processes.

4.2.NodeJS in developing social network:

As we have defiend, NodeJS is the runtime environment that allows us JavaScript to build front-to-back web applications within just a single programming language: JavaScript. It triggers a renaissance JavaScript and boosts it to become one of the most common language in use nowaday. NodeJS is built on the V8 engine that was included in Google Chrome browser. It is desinged to aim for process loop events and emphasises asynchronous, with non-blocking I/O. Most importantl thing, it is suitable to building scalable network applications such as social networks or dynamic applications with variety of social features.

One the process that NodeJS best in handling is real-time processing, it enable us to build, scale and personalize users news feeds or activites strreams. No matter that is aggregated feeds, ranked feeds, real-time messaging or push notifiication, NodeJS can handle them all.

And one more reason why I choose NodeJS to develop social network is the development speed and community support. To accelerate development on NodeJS runtime engine, several of popular libraries and web frameworks have emerged with it. With the powerful package manager NPM, which has appeared at the early of NodeJS development, developers community has tended towards smaller, more focused on libraries than versatile frameworks. As a result, it is much more common to integrate multiple single-purpose libraries when building with NodeJS.

4.3.Express framework:

Express (or Express.js) is a NodeJS based web application framework that designed focus to minimalism and flexibitiy, which provides us a powerful collections of features for building web

applications services and mobile apps. Additionally, Express provides a middleware layer of the most common fundamental web application features, but still leverage the NodeJS strength.

The reason for choosing Express to implement this social network project is because its simple and flexibility. With lots of HTTP utility methods and middleware that cover all of our disposal, it is quick and easy to build a back-end data API. Moreover, it is also easy to itegrate the created API with GraphQL libraries and resources such as Apollo Server.

5. ReactJS – Perfect choice for GraphQL client-side

5.1.A brief introduction to ReactJS:

ReactJS (or just React) is a JavaScript library for building user interfaces. It is mainataned by Facebook and public to open-source for community collaboration of developers and tech companies. React has been developed based on the idea of gradual adoption, which meant we can use it as simply a library to create addition interactive to render data to the DOM (Document Object Model) or implement a complex powerful React-application. Howevers, because React only concern to DOM rendering, so, in order to create React-applications, we often required to use additional libraries for state management and navigation such as Redux or React-Navigation.

5.2.Major advantages of ReactJS:

First of all, the **declarative**. ReactJS is extremly easy to create interactive user interfaces. All we need to do is design the simple views for each of state declared in our application, then wait for the magic. To be more clearly, React can automatically and efficiently update or render right exactly components that match the data of states when it change. Thanks to that, our code could be more predictable and simply for debugging. Secondly, React is based on **component-based** module architecture, which meant React allows us to build encapsulated components that is able to independently manage their state, then compose them create dynamic user interfaces. Once the

logic components is created in JavaScript instead of web-template (like Bootstraps), developers could easily pass data through app by states system and keep those states out of the rendered HMTL DOM. Finally, **write once and build anywhere**. ReactJS do not assumpt about any other part of project ‘s technologies and tools, developers now can develop new features in any react applications without rewrite what they have done before. Morovers, React can both render on web based platform through Node runtime engine and power mobile applications with React-Native framework.

5.3.Understand component in React:

In React, each component construct a **render()** method, which collects input data then returns whatever views we are about to display. The following example use a syntax called JSX to create a returned display view.

```
class Greeting extends React.Component {
  render() {
    return (
      <div>
        Hi {this.props.myMame},welcome to React JS !
      </div>
    );
  }
}

ReactDOM.render(
  < Greeting name="Minh" />,
  document.getElementById('greeting-sample')
);
```

As we can see, the input data myMame is passed into Greeting component and could be accessed by render() method by this.props . The we could get something like this in the browser view.

Hi Minh, welcome to React JS !

Howevers, JSX is nothing more an option and is not required in React-component. The above JSX syntax code could be converted to traditional XML as following:

```
class Greeting extends React.Component {
   render() {
     return React.createElement(
       "div",
       null,
       "Hi",
       this.props.myMame
     );
   }
 }

  ReactDOM.render(React.createElement(Greeting, { myMame: "Minh" }),
document.getElementById('greeting-sample'));
```

Aditionally, a React-component could maintain internal data as states in order to take input data through this.props. Whenever each state of component change their data, the rendered display could be triggered to update by re-call the render() method. Here is an example about creating a **stateful component** that render a counting timer that count in seconds the time that view is displayed, written in JSX syntax.

```
class Timer extends React.Component {
   constructor(props) {
```

```
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(state => ({
      seconds: state.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(
  <Timer />,
  document.getElementById('timer-example')
);
```

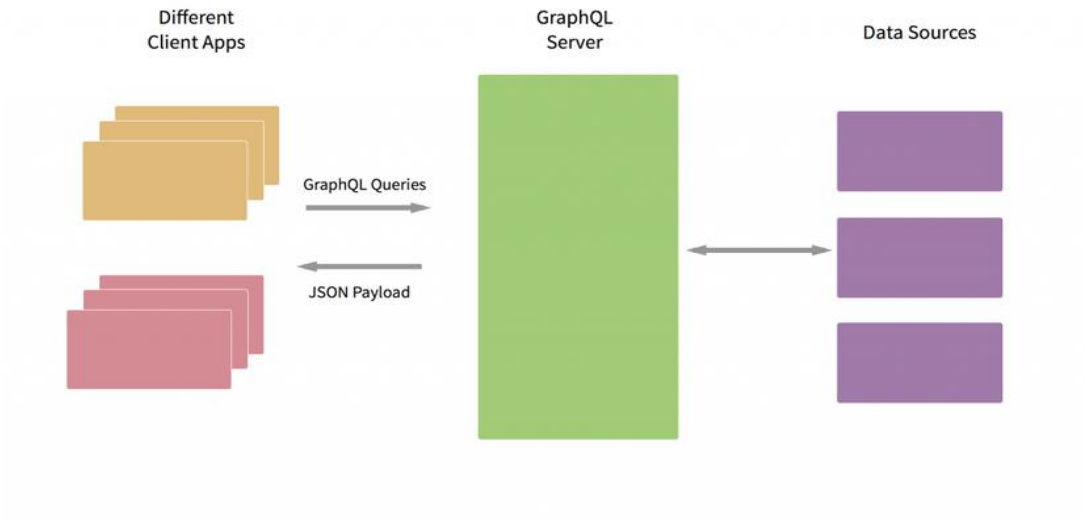
Chapter III: Technologies and Tools

1. Apollo Server

1.1. What is Apollo Server?

Apollo server is one server based on GraphQL, and it can compatible with many GraphQL Client, so it is the best way to build a GraphQL API, which can use data from many sources. By that definition, we can see the are many advantages of Apollo. We can use as one stand-alone GraphQL server, or we can use it as one environment without server (or serverless). If we want, we can use GraphQL server as one add-on for our node.js middleware application. And we can also use it as the gateway, which can use for federate data graph.

What Apollo server can provide to us? We can start to fetching data quickly because it can setup easily. Due to the incremental adoption, we can add new features whenever we need. Apollo can also compatible with any data source, build tools and GraphQL client (as the definition said, because it is based on GraphQL). And Apollo can enable us to create and deploy our features faster than before.



1.2. Data manipulation in Apollo Server:

a. Resolvers:

Data resolver is one function which take responsible to populate data in our schemas. Our jobs are about define the data, and then the resolver fetching all the data from where ever we want (back-end database or third-party API).

One good thing of Apollo is, if we forgot (or don't want) to define resolvers for one field, the Apollo server will do it for us, it will define one default resolver for that field, by this way, it can reduce loss data.

b. Data source:

Data source is one class that can fetching data from other service. Apollo have the built-in support for this class, so we just need only write the back-end code, and watching Apollo do all the work for us. This class will reduce the code we need to write, we can use that time to do others thing, like improve some feature, or write new feature.

c. Error handling

Apollo provides us a lot of pre-defined errors, like "AuthenticationError", "ForbiddenError', "UserInputError". It can help our debugger easier to debug, allow client to do some action based on error we have catch.

d. File uploads

Apollo allow us to upload the file by default if we use some server integrations which support file upload (for example: Express, hapi, Koa, …) we can enable uploads file by references "upload" types in the schemas passed to Apollo server.

e. Subscriptions

Subscriptions are one GraphQL operation which is tracking the event from Apollo server. The good thing is, we can have the GraphQL subscriptions without any configuration, because as I say, Apollo is based on GraphQL, and it can help us a lot because we don't need to configuration too much.

1.3. Performance and security:

It is all about the data fetching. How about the performance support for Apollo? It is the caching system. Apollo server provide to us "one mechanism for author of severs to declare fine-grained cache control parameters" (reference) for every GraphQL types and fields. For each request sent to server, the Apollo server will collect all the hints from all field of executed query and use it to make caching for feature. This caching style is made project use Apollo server have the better performance than others server. But the better performance cannot make the project more secure, let take a look about the security method of Apollo server to understand and find out does it have the good security method.

The security method in Apollo is about the authentication. Have you ever wondered what if your data can be read and edit by other people do not have the permission? Authentication is the answer for this question. This method will determine if user logged into our system, and determine who are they, does they have the permission to access the data of website. And by that way, we can decide what that user can do with our data (of course decide by the developed code, because we cannot decide it manually every time someone logged in our server).

2. Prisma

2.1. What is Prisma?

Prisma is one open-source database toolkit, support for Apollo, which I have introduce right above this section. Prisma is consists 3 following parts: Prisma client, Prisma migrate and Prisma Studio.

The Prisma Client can use for any node.js or typescript back-end application, it very fit with my project, so I choose Prisma as database toolkit. But if only that, it will not the reason why I choose Prisma, I choose Prisma because it has the advantage with others database language. So, what is the advantage and why I choose Prisma instead of SQL or ORMs? I will show you next.

2.2. Advantages when using Prisma:

First advantage is Prisma can help us if we get problem with SQL, ORMs and other database tools. If I choose raw SQL, I will have one database tool with full control about database concept, but low productivity. Why? Because sending SQL simple string to database can slow down the performance of my project (raw SQL not have good connection handling, everything is manual). And raw SQL is not safe for my query, I cannot send my query by using raw SQL without concern about the data hacking or SQL Injection, so raw SQL is not the good choice, in any case. And what happen if I choose ORMs? The ORMs look like the good choice if we compare it with raw SQL: less control to database but more productivity. What is the point? The point is ORMs use the mapped to objects, but the data in SQL is table, so it can cause the Object-relational impedance mismatch, if you keep working with ORMs, maybe you can cause many and many errors when you running your code, as developer, I don’t like it.

Second advantage is about the data and SQL queries. Our developer should care about the data, not the SQL query, and Prisma can do it very good. Our developer should care about the data, and make the feature complete as soon as possible, not keep going on doing SQL query again and again to get the result meet their requirement. The power of SQL is one good thing, but even you are having a 10, 20-year experience with SQL, you can still get the mistake, and debugging phase will take a lot of time. Do you want to debug 1 week to find the bug, or you want complete it on 1 days by using Prisma, and use 6 days later to find out how you can create new feature and deploy it?

And the last thing, Prisma can make developer more productive. The main reason of Prisma is help developer have more time to develop other feature than working with database. Prisma is using object, not like ORMs try to mapping with data, they use the object for database. It also avoids the complex model objects, because the queries are not classes. Prisma also have auto-completion in the editor, so you do not need to look documentation every time you want to create one new database or using new queries for some feature. And Prisma database queries can also validated when you compile your code, no more fixing database bugs in many days.

2.3. How Prisma work?

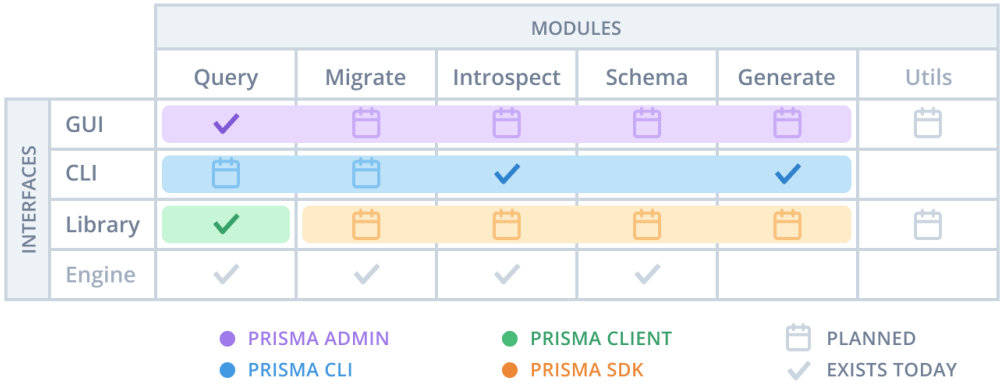
That is the advantage if compare with other database toolkit, so how does Prisma work? The Schema: every project use Prisma as database toolkit will start with one file of Prisma schema. With this file, you can define the application models in one intuitive data modeling language, and you just only need to configure 3 things: data source (make connection to your database), Generator (confirm that you want to use one Prisma Client and it will auto generated), and Data model (the model of your application, you need to define it on Prisma schema file).

There are many supported libraries and frameworks that you can use with Prisma: Express, koa, hapi, Fastify, Sails, AdonisJS, node.JS, … and it is fit with my project.

- **Data modeling on Prisma:**

You can use 2 workflows for data modeling: only Prisma client and using Prisma client with Prisma Migrate. No matter which you use to approach, Prisma will never create application models in your programming by define class, interface or structure manually, it is defined in your Prisma schemas file.

- **Understand Prisma:**



(source: prisma.io)

Modules: Prisma is provides some feature, and it is form as modules. I will review these features:

- Query: this module will send execute query to your database, just like any database toolkit will do
- Migrate: make the schemas migrations in your database system
- Introspect: work like compiler of Prisma, it will translate database schema into the data model of Prisma data toolkit
- Schema: build, and modify the Prisma schema file, which is defined in your code
- Generate: generate one client for becoming data source of Prisma

Interface: like its name, it will provide you the interface to use the modules of Prisma, with 4 level of interface:

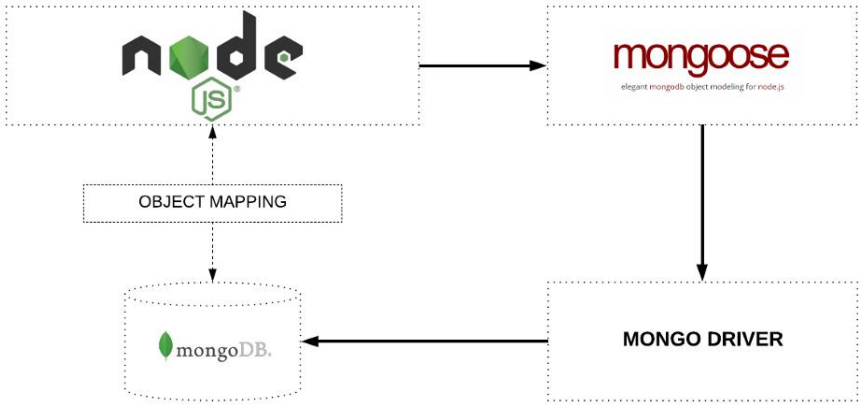
- Engines: Engine will implement core functionality, which is used by other level interfaces (in this case is higher – level)
- Library: Library will help programmer can call the module to their code and use it
- CLI: CLI is one way to use module function on any command line
- GUI: GUI is one visually way to use module function

Prisma engine: Prisma engine is the “direct interface to the database”. The engine layer will take responsible to make the communicate between any higher – level interface with database.

3. Mongoose

3.1. Brief introduction to Mongoose:

Mongoose is an Object Model Mapper (ODM) library for MongoDB and NodeJS. It manages relationships between data, provides schema validation, and is used to translate objects in code and represent objects in MongoDB.



Mongoose model includes Mongoose schema. Mongoose schema defines the structure of the document, the default values and validation. while Mongoose model provides an interface for databases to create, query, update, delete records.

3.2. Data modeling in Mongoose Schema:

Creating a Mongoose model consists of mainly three parts:

- First, import Mongoose to your file:

```
let mongoose = require('mongoose')
```

This import will be the same as the one returned when we connected to the database, meaning that the schema and model definitions will not need to be explicitly connected to the database.

- Second, define a schema properties through an object whose key name corresponds to the attribute name in the collection.

```
let usernameSchema = new mongoose.Schema({
   username: String
 })
```

Here we define an attribute called email with a String schema type mapping to an internal validator that will be triggered when the model is saved to the database. It will fail if the data type of the value is not a string type. The following types of schemas are allowed:

Array Boolean Buffer Date Mixed Number ObjectId String

Mixed data type might be a generic or flexible data type. Both Mixed and ObjectId are defined as required ('mongoose') Schema.Types.

- Third, define a schema properties through an object whose key name corresponds to the attribute name in the collection.

```
module.exports = mongoose.model('Username', usernameSchema)
```

Let's combine the above code to ./src/models/username.js to define the content of a basic model:

```
let mongoose = require('mongoose')
let usernameSchema = new mongoose.Schema({
  username: String
})
```

```
module.exports = mongoose.model(‘Username’, usernameSchema)
```

The schema definition must be simple, but its complexity is often based on application requirements. Schemas can be reused and they may also contain some sub-maps. In the example above, the value of the email attribute is a simple value type. However, it can also be an object type with additional attributes on it. We can create an instance of the model we have defined above and populate it using the following syntax:

```
let UsernameModel = require('./username')
let res = new EmailModel({
  username: 'minhnhgcs16319'
})
```

Enhance the Username schema to make the email attribute a required field, and convert the value to lowercase before saving it. We can also add a validation function that will ensure that the value is a valid email address. We will refer and use the browser library has been installed before.

```
let mongoose = require('mongoose')
let validator = require('validator')
let usernameSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true,
    validate: (value) => {
      return validator.isEmail(value)
    }
  }
})

module.exports = mongoose.model('Username', usernameSchema)
```

Basic operation Mongoose has a flexible API and provides many ways to accomplish a task. We will not focus on this part as it is outside the scope of this article, but keep in mind that most operations can be done in a variety of ways using different syntaxes or through application architecture.

3.3. Query building in Mongoose Schema:

Mongoose has a very rich API to handle the many complex operations supported by MongoDB.

```
PostModel.find()                    // find all posts
        .skip(50)                   // skip the first 100 posts
        .limit(8)                   // limit to 8 posts
        .sort({authorName: -1}      // sort descending by authorName
        .select({category: true}    // select category only
        .exec()                     // execute the query
        .then(docs => {
          console.log(documents)
        })
        .catch(error => {
          console.error(error)
        })
```

3.4. Advangtages of using Mongoose Schema:

We have only explored some of Mongoose's basic features. This is a rich library full of useful and powerful features when working with data models in the application layer. You can interact with Mongo directly with Mongo Driver, Mongoose will simplify that interaction by allowing you to model the relationships between data and validate them easily.

4. JSON Web Token

4.1. What is JSON Web Token?

JSON Web Code (JWT) is an open standard (RFC 7519) that defines a compact and closed way to securely transmit information between parties as JSON objects. This information can be verified and trusted because it contains a digital signature. JWTs can be signed with a secret algorithm (with HMAC algorithm) or a public / private key using RSA encryption.

An example of JWT Token:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiZW1haWwiO
iJtaW5obmhAdW5pdC5jb20udm4iLCJpYXQiOjE1ODg0NzQ5MTEsImV4cCIQ
MTYyMDAzMjUxMX0.Rc5qOaSGVgmlqgTK7IXV6p7q2zh4VesOw0SsC6dZbk
```

At first it looks complicated, but if understood, the structure of a JWT is as follows:

```
<base64-encoded header>.<base64-encoded payload>.<base64-encoded signature>
</base64-encoded>
```

In other words, JWT is a combination (by the sign.) An Object Header in JSON format encode base64, a payload object in JSOn format is encode base64 and a Signature for URI is also base64 encoded as well.

4.2. The 3 components of JWT:

- **Header:**

The header consists of two main parts: the token type (the default is JWT - This information indicates that this is a JWT token) and the algorithm used for encryption (HMAC SHA256 - HS256 or RSA).

```
{
  "alg": "HS256",
  "typ": "JWT"
 }
```

- **Payload:**

The payload contains claims. Claims are expressions of an entity (such as a user) and some auxiliary metadata. There are 3 common types of claims in Payload: reserved, public and private claims. Reserved claims: These are some predefined metadata, some of which are required, others should be followed for JWT to be valid and well-informed:

iss	issuer
iat	issued-at time
exp	expiration time
sub	subject
aud	audience
jti	JWT Unique Identifier (prevent JWTs from replayed, helpful for one time tokens)

Example for a JWT payload:

```
{
  "iss": "jira:1413309",
  "iat": 1502812350,
  "exp": 1509812390,
  "qsh": "dd0f8063ff4ca797b4f7f6207d4922e6d1c66ea6bc90c8ab1e41c614b719ad79",
  "sub": "spiderman",
  "context": {
     "user": {
        "userKey": "spiderman",
```

```
        "username": "petterP",
        "displayName": "Peter Parker"
      }
   }
}
```

Public Claims - Claims are widely recognized and used by the community.

Private Claims - Self-defined Claims (not identical to Reserved Claims and Public Claims), created to share information between the two parties agreed and agreed upon earlier.

- **Signature:**

Signature Signature in JWT is a string encrypted by header, payload along with a secret string according to the following principles:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Because the Signature itself includes both headers and payloads, the Signature can be used to check the integrity of the data when transmitting.

4.3. When to use JWT?

One of the common JWT application situations is:

Authentication: The most common scenario, when the user is logged in, every subsequent request is accompanied by a JWT token chain, allowing users to access the paths, services and resources allowed to correspond to the token. Single Sign On is also a function that uses JWT widely, because the JWT string is small enough to attach to the request and used in many systems of different domains.

Information Exchange: JSON Web Token is also an effective and secure way to exchange information between multiple applications, because JWT must be signed (using public / private key pair), you can make sure that the sender is the one which they say they are (more abstractly than not or difficult to impersonate with JWT), in addition, the signature is also calculated based on the content of the header and payload content, so that you can authenticate The content is original, has not been edited or tampered with. However, a very important note is that due to the simple structure of JWT, JWT can easily decode, so you should not use JWT to transfer sensitive information.

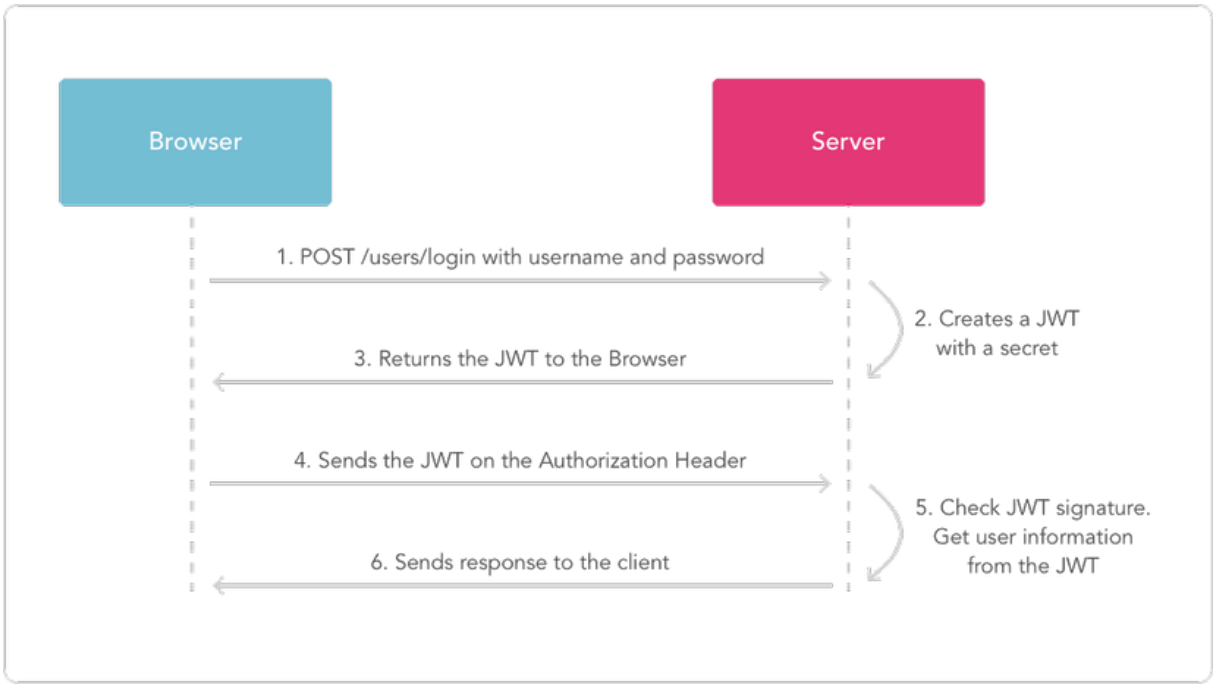
4.4. How does JWT work?

Here, I am specific example of the application of JWT in the Authenticate problem (authentication). In authentication, when the user logs in successfully (Browser will post the username and password to the Server), the Server will return a string JWT on Browser, and this JWT Token needs to be saved in the user's Browser (usually LocalStorage or Cookies), instead of the traditional way of creating a session on the Server and returning Cookies.

Whenever the User wants to access a protected route (only logged in users are allowed), the Browser will send this JWT token in the Header Authorization, the Bearer schema of the request sent.

```
Authorization: Bearer <token>
```

This is how stateless authentication works, the state of the user is not stored in the server's memory but is fully packaged into JWT. The server will check if the JWT Token is valid or not (Because JWT is self-contained, all information needed to check the JWT is already contained in the JWT Token).



Because of the stateless nature, we no longer have to worry about which domains are used for your API, as there is no more trouble with CORS (Cross-Origin Resource Sharing) because it doesn't use cookies.

Chapter IV: Software Product Requirement

1. Functional requirement

Our social media app makes easy to connect and share with family and friends online, is software that allows users to share photos add captions, edit filters, tweak settings, engage with others (like, comment) explore to find other users.

No	Role	Feature
1	User	-Register -Login -Search user account -View profile -View notification -Update profile -Add new post -Read message and start a chat -Search user who sent message -View newsfeed with image -Explore new user

2. Software and Hardware requirement

Operating System: Windows 7+, Linux Kernel 2.4 +, MacOS 10.13 High Sierra.

- Git (version 2.26 +): Cloning and controlling source code.
- Nodejs (version 10.14+): JavaScript runtime environment.
- Yarn (version 2.x +): Package manager.

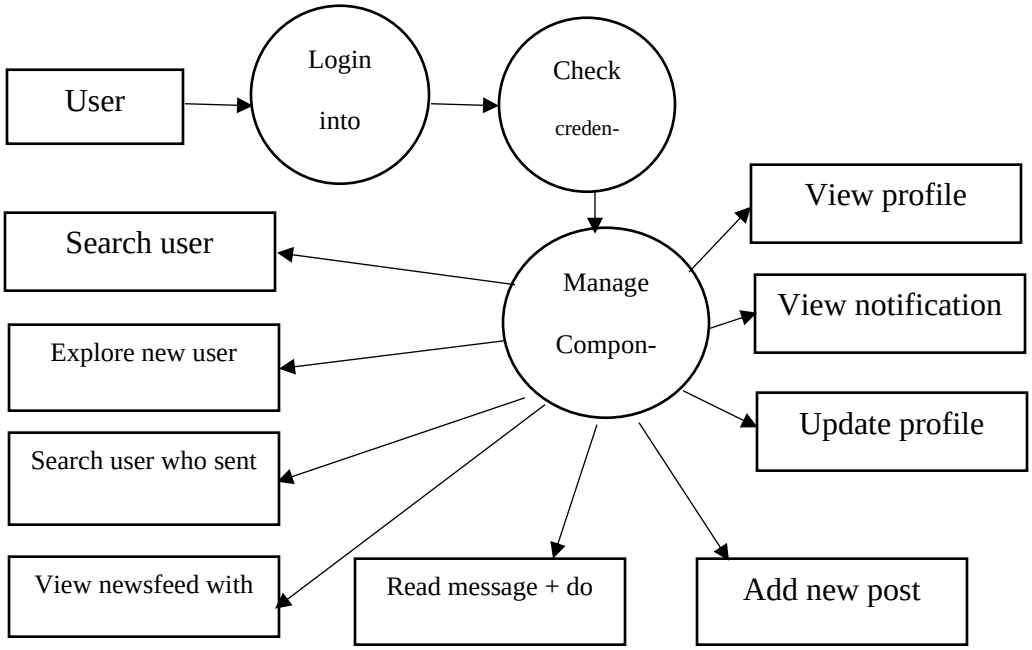
Tools:

- Database management tools: Navicat Premium 12 or TablePlus (recommended).

- Code Editor: Atom, SublineText or Visual Studio Code(recommended).

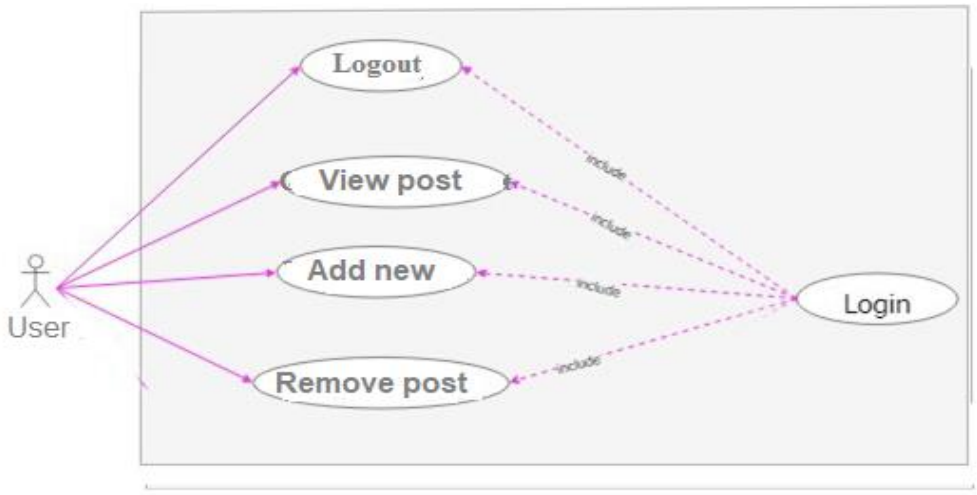
Minimum Ram: 512 MB + - Free space: 500 MB+

3. Data flow diagram



4. List of use-cases

4.1. Posts management



USE CASE – 001			
Use Case No.	**001**	**Use Case Version**	2.0
Use Case Name	User's role		
Author	Minh Nguyen		
Date	01/05/2020	**Priority**	Normal

Actor:

- User

Summary:

- This use case describes role of user.

Goal:

- Allow managing post.

Preconditions:

User haven signed into system.

Post Conditions:

- **Success:** User is authorized, redirect to corresponding page.
- **Fail:** System will show error messages on current page.

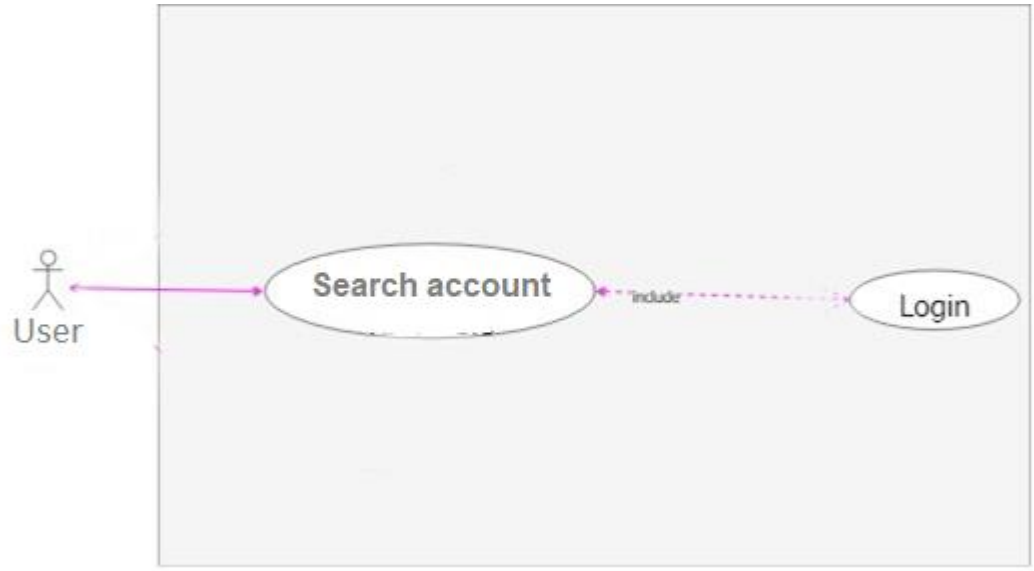
Main Success Scenario:

Step	Actor Action	System Response
1	*View user 's posts* – Click on view my timeline	System redirect to user 's page that contain user 's order by created datee.
2	*Choose to create post*– Click on Add New Post	System focus to post create area and blur out others.
3	*Input post content* – Input post content and click on Photo to upload image	System render a preview for creating post

4	*Submit post* – Click Share to upload post to time line	System add new created post to first of timeline
5	*Delete post* – Click on post option (triple dots icon on top right corner) and choose delete post	System remove post from timeline

Relationships: N/A
Business Rules:
- Only active account can login into the system.

4.2. Search users:



USE CASE –001			
Use Case No.	**002**	**Use Case Version**	2.0
Use Case Name	User's role		
Author	Minh Nguyen		
Date	01/05/2020	**Priority**	Normal

Actor:
- User

Summary:
- This use case describe user's role.

Goal:
- Allow allows administrator to edit his profile.

Preconditions:
User haven signed into system.

Post Conditions:

- **Success:** Admin is authorized, redirect to corresponding page.
- **Fail:** System will show error messages on current page.

Main Success Scenario:

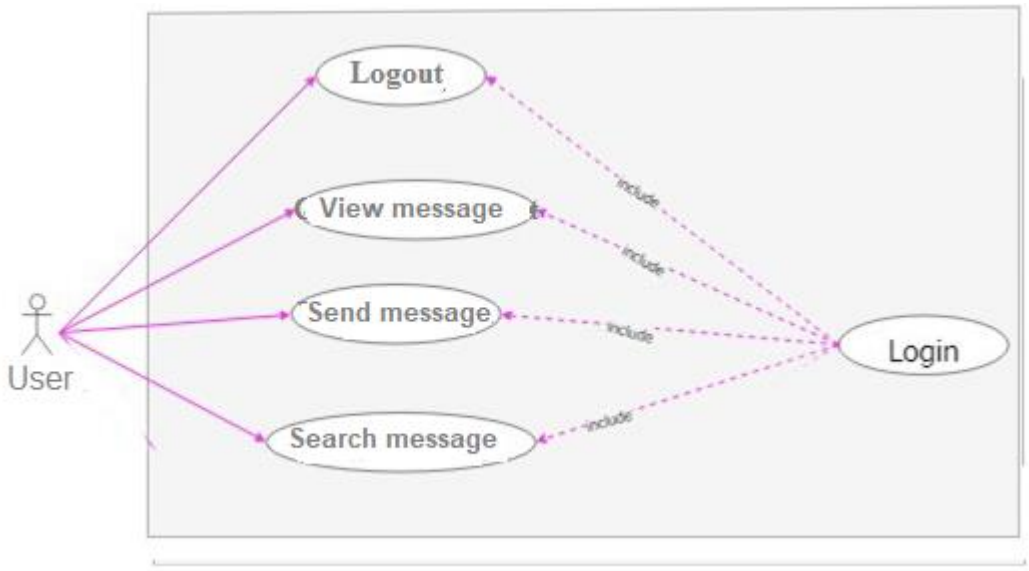
Step	Actor Action	System Response
1	*Choose to search user* – Click on Search Bar	System focus to search bar text input
2	*Input search keyword* – Input keyword to text input	System rerturn a drop down list of user that match the input key word
3	*Navigate to searched user time line* – Click on a item of dropdown list	System redirect to chosen user timeline

Relationships: N/A

Business Rules:

- Only active account can login into the system.

4.3. Message management



USE CASE –001			
Use Case No.	**003**	**Use Case Version**	2.0
Use Case Name	User's role		
Author	Minh Nguyen		
Date	01/05/2020	**Date**	01/05/2020
Actor: - User			

Summary:

- This use case describe user's role.

Goal:

- Allow allows administrator to edit his profile.

Preconditions:

User haven signed into system.

Post Conditions:

- **Success:** Admin is authorized, redirect to corresponding page.
- **Fail:** System will show error messages on current page.

Main Success Scenario:

Step	Actor Action	System Response
1	*View conversation* – Click "Messages" link on side bar	System redirect to Messages management view
1	*View conversation* – Click "Messages" link on side bar	System redirect to Messages management view
2	*Find user to message* – Input keyword to find user at "To:"	System response list of users that match the keyword
3	*Start conversation* – Click on user in the list	System render conversation view
4	*Send message* – Input message to text input at the bottom of conversation view. Then click send button	System display new message in the conversation

Relationships: N/A

Business Rules:

- Only active account can login into the system.

4.4. News feed action

USE CASE –001			
Use Case No.	**004**	**Use Case Version**	2.0
Use Case Name	User's role		
Author	Minh Nguyen		
Date	01/05/2020	**Date**	01/05/2020

Actor:

- User

Summary:

- This use case describe user's role.

Goal:

- Allow allows administrator to edit his profile.

Preconditions:

User haven signed into system.

Post Conditions:

- **Success:** Admin is authorized, redirect to corresponding page.
- **Fail:** System will show error messages on current page.

Main Success Scenario:

Step	Actor Action	System Response
1	*Load more posts*– Scroll list to end	System load more posts to news feeds
1	*Comment to post* – Input comment and submit	System render comment to post
2	*Like post* – Click on thumb up icon	System change post behaviour to liked
3	*delete comment* – Press delete icon next to comment	System remove comment from post
4	*Unlike post* – Click on thumb up icon again	System change post behaviour to unliked

Relationships: N/A

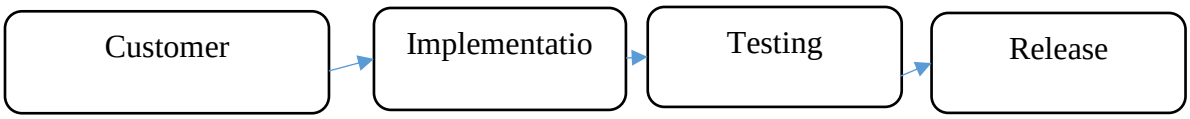
Business Rules:

- Only active account can login into the system.

Chapter V: Review of Software Development Methodologies

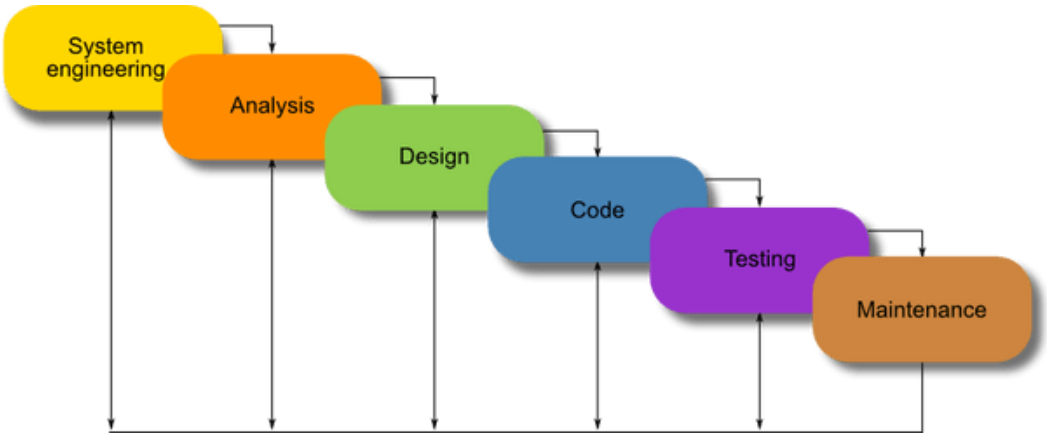
Software development life cycle is the time from when a customer proposes an idea to when the software is handed over and put into use.

Depending on the model applied, the division of phases may vary. Basically, there will be the following steps:



1. 'Waterfall' model:

The waterfall model is one of the most easily understood and manageable project management models available today. The waterfall model is a project management method based on sequential and sequential design process. The stage is only started when the previous stage has been completed.



Specifically:

Stages	Activities	Output

Requirement analysis	Survey, collect customer requests	Requirement specification
Design	Implement overall design, detailed design	Database, design document
Implementation	Perform coding according to design documents	Source code
Testing	-Create the test script. -Perform the test -Update the test results -Error handling	Test case and errors
Deployment	Deploy the application on real environment	Deployment report with customers
Maintenance	Maintain software upgrades based on problems detected during use	Software

Getting the requirements right from the start is a challenge. In one of the first stages of the project. The biggest drawback of Waterfall is how to manage changes because customers don't really know what they need. The team cannot switch between stages, even when a change occurs. When the project goes to the test stage and discovers a lack of a function in the customer's request, returning to fix it is very expensive and difficult to perform. The product is delivered late: The project will need to complete 2 to 4 stages before the actual coding starts, so the stakeholders will have to wait until the final stages to see the product.

2. RAD & Agile model:

Nowadays, software creation is extremely complex. The increasingly complex algorithms are difficult to build and manage. Therefore, software managers have been constantly learning and

searching to create better software development methods as traditional models have had a hard time responding to user changes.

In the prototype, the process begins by collecting requests in the presence of representatives from both the developer and the customer in order to determine the overall goal of the software system. Then, implement a quick design that focuses on conveying aspects through the prototype so that customers can visualize and evaluate to fulfill the requirements for the entire software system. This not only helps refine the requirements, but also helps the development team to better understand what needs to be developed. Following the prototype phase can be a waterfall model cycle or can be another model. Prototype is a working model of software with a limited number of functions. Prototypes do not always hold the exact logic used in actual software applications. This prototype was developed based on currently known requirements. The prototype is a software development model.

Agile is a flexible software development model, based on iterative and incremental methods. It will engage customers in the development process of the software, people try to produce the product as quickly as possible. Then give the customer a trial and feedback, the development team will continue to develop the next stage. Depending on the project, the time to release the product is long or short (approximately two weeks, or up to one month)

In software projects we will have difficulty in collecting fully and accurately good customer requirements from the beginning. There are so many issues that affect software development, and so many things that we don't anticipate come from factors such as business, technology, people and so on.

Rapid Application Development is a form of agile software development methodology, although RAD and the agile methodologies share similar values, the differences are pretty obvious.

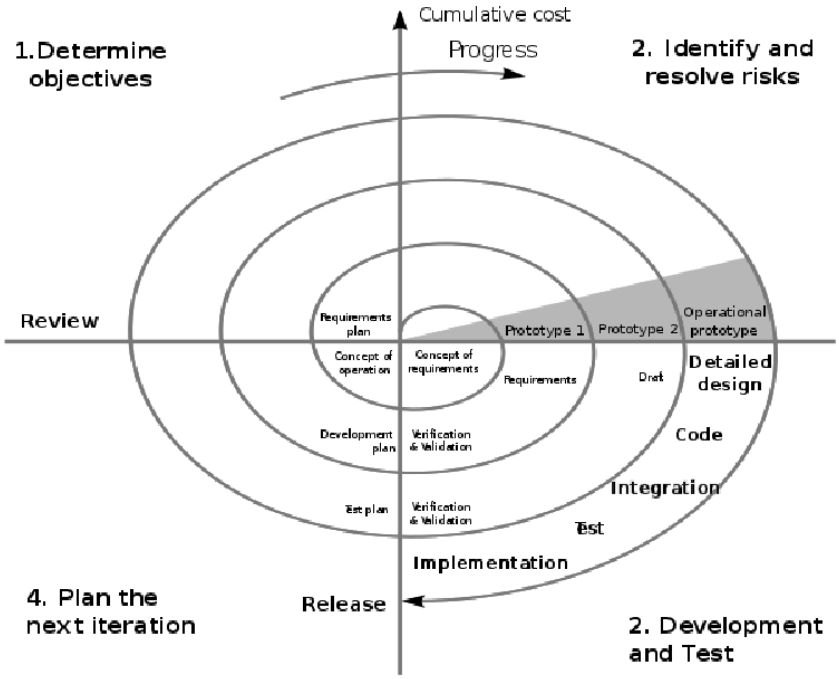
RAD	Agile
Design prototype	Design un-finished work
Include all technical team	Include non-technical team (experienced user)
Managed by project manager	Self-management, led by customer
Individuals communication	Teamwork communication

3. Spiral model:

The spiral model can be considered a combination of a waterfall model and a prototype model and also adds a risk assessment.

In the spiral model, the software development process is represented as a spiral. The phases in the spiral development process include:

- *Goal setting*: define goals for each phase of the project.
- *Risk assessment and mitigation*: risk is assessed and actions are taken to minimize the risk.
- *Development and evaluation*: after risk assessment, a system building model will be selected from the common model.
- *Planning*: project evaluation and the next phase of the spiral model will be planned.



The idea of reducing risk through the use of new panels and a number of construction tools for other buildings to bring a new model: a spiral model. The most important way to look at this model is the waterfall model in which each phase is to be added to the risk analysis. Before starting a certain phase, people analyzing the risk at the beginning and look at how it can be solved. If there is no way to describe how to deal with important risks, the project will end.

4. Software development methodology selection:

This project applies a V-model, which is an extension of the waterfall model. It has a corresponding testing phase for each development stage. Therefore, for every stage of the development cycle, there is a relevant test phase. The corresponding test phase of the development phase is planned in parallel. This model is also known as verification and verification model.

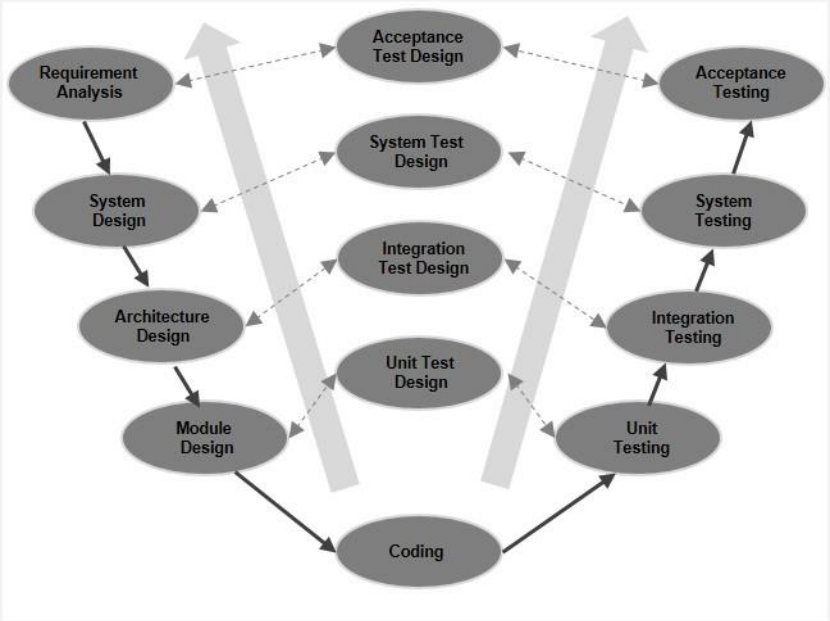
Advantages:

- Simple, easy to use.
- There are activities, specific plans for the testing process.

- Save time, and have a higher chance of success than the waterfall.
- Be proactive in bug detection, find bugs at the beginning.

Drawback:

- Difficult to control risks.
- Not a good model for complex and object-oriented projects.
- The model does not work well for long and ongoing projects.
- Not suitable for projects with risk of changing requirements.



Chapter VI: Design and Implementation

1. Database design

1.1.Table Message

Column	Data type	Constraints
createAt	datetime(0)	
id	int(11)	Primary key
message	varchar(191)	
receiverID	int(11)	
seen	tinyint(1)	
senderId	int(11)	
updatedAt	datetime(3)	

1.2.Table User

Column	Data type	Constraints
coverImage	varchar(191)	
coverImagePublic	varchar(191)	
createdAt	datetime(0)	
email	varchar(191)	
fullname	varchar(191)	
id	int(11)	Primary key
image	varchar(191)	
imagePublicID	varchar(191)	

isOnline	tinyint(1)	
password	varchar(191)	
passowrdResetToken	varchar(191)	
passwordResetTokenExpiry	datetime(3)	
updateAt	datetime(3)	
userId	int(11)	
username	varchar(191)	

1.3.Table Migration

Column	Data type	Constraints
revision	int(11)	Primary key
name	text	
datamodel	longtext	
status	text	
aapplied	int(11)	
rolled_back	int(11)	
datamodel_steps	longtext	
database_migration	longtext	
errors	longtext	
started_at	datetime(3)	
finished_at	dateime(3)	

1.4.Table Comment

Column	Data type	Constraints
authorID	int(11)	
comment	varchar(191)	
createAt	datetime(0)	
id	int(11)	Primary key
postID	int(11)	
updateAt	datetime(3)	FK

1.5.Table Post

Column	Data type	Constraints
authorId	int(11)	
createAt	datetime(0)	
id	int(11)	Primary key
image	varchar(191)	
imagePublicId	varchar(191)	
title	varchar(191)	
updateAt	datetime(3)	FK

1.6.Table Like

Column	Data type	Constraints
createdAt	datetime(0)	
id	int(11)	Primary key

postId	int(11)	
updateAt	datetime(3)	FK
userId	int(11)	

1.7.Table Notification

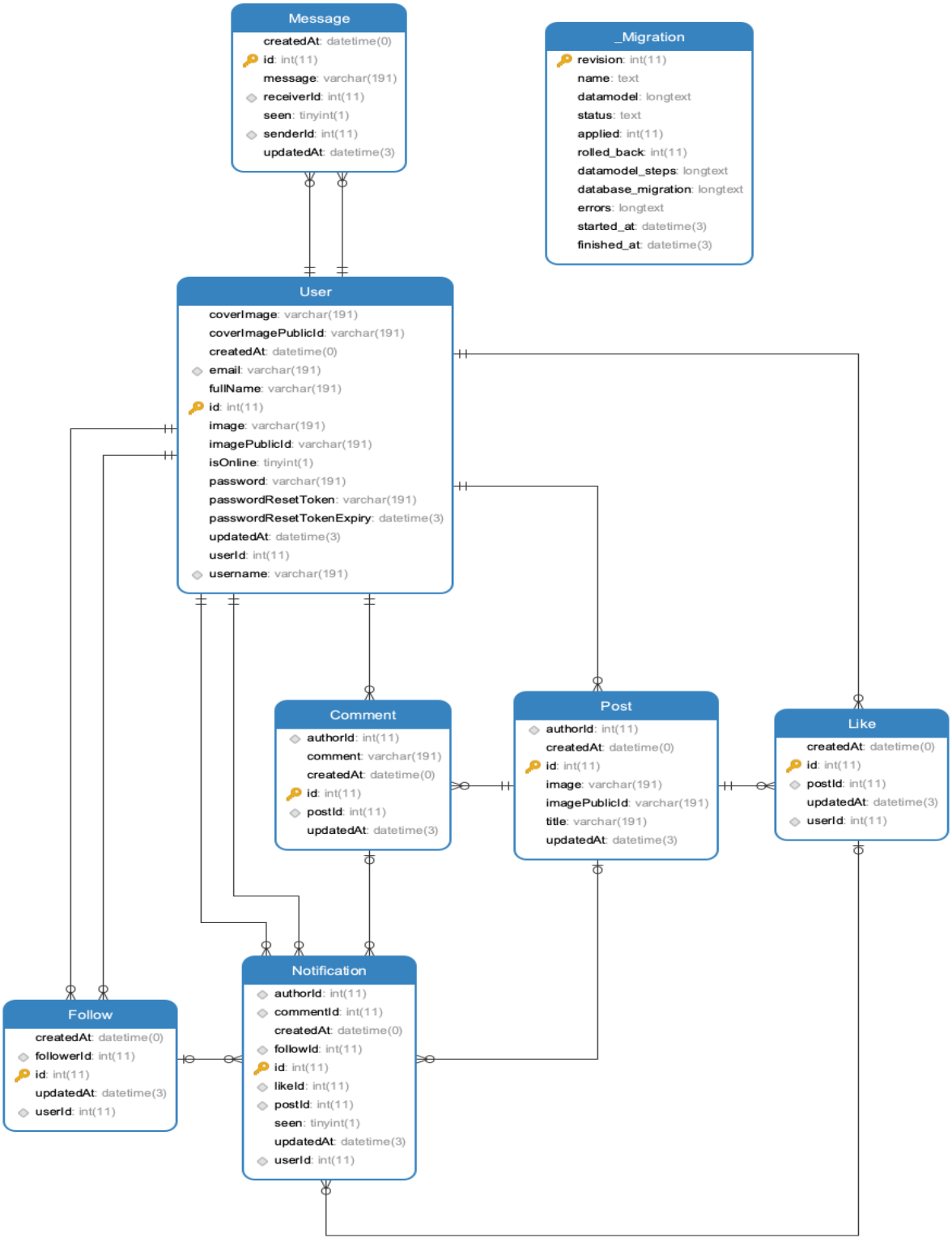
Column	Data type	Constraints
authorId	int(11)	Primary key
commentId	commentId(11)	
createdAt	date(0)	
followId	int(11)	
id	int(11)	
likeId	int(11)	
postId	int(11)	
seen	tiny(1)	
updateAt	datetime(3)	
userId	int(11)	FK

1.8.Table Follow

Column	Data type	Constraints
createAt	datetime(0)	
followId	int(11)	
id	int(11)	Primary key

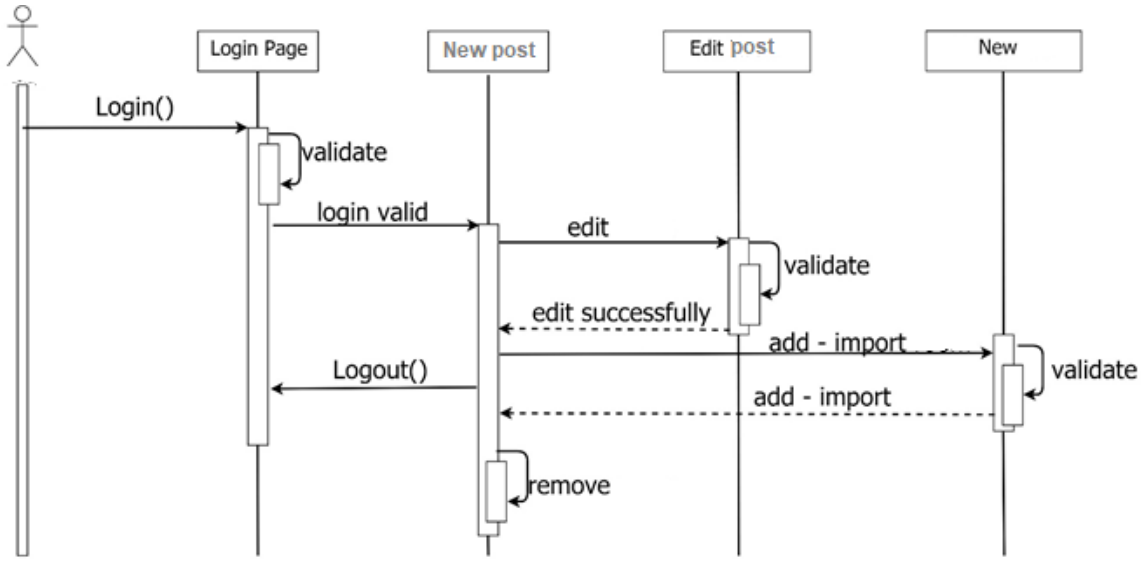
updateAt	datetime(3)	
userId	int(11)	FK

2. Entity rational diagram

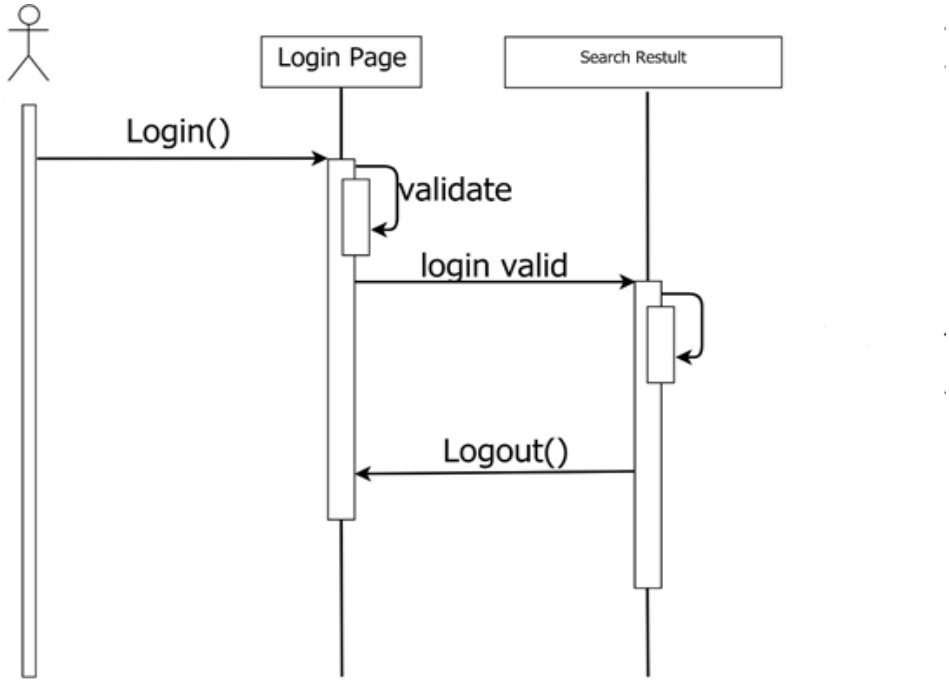


3. Sequence diagram

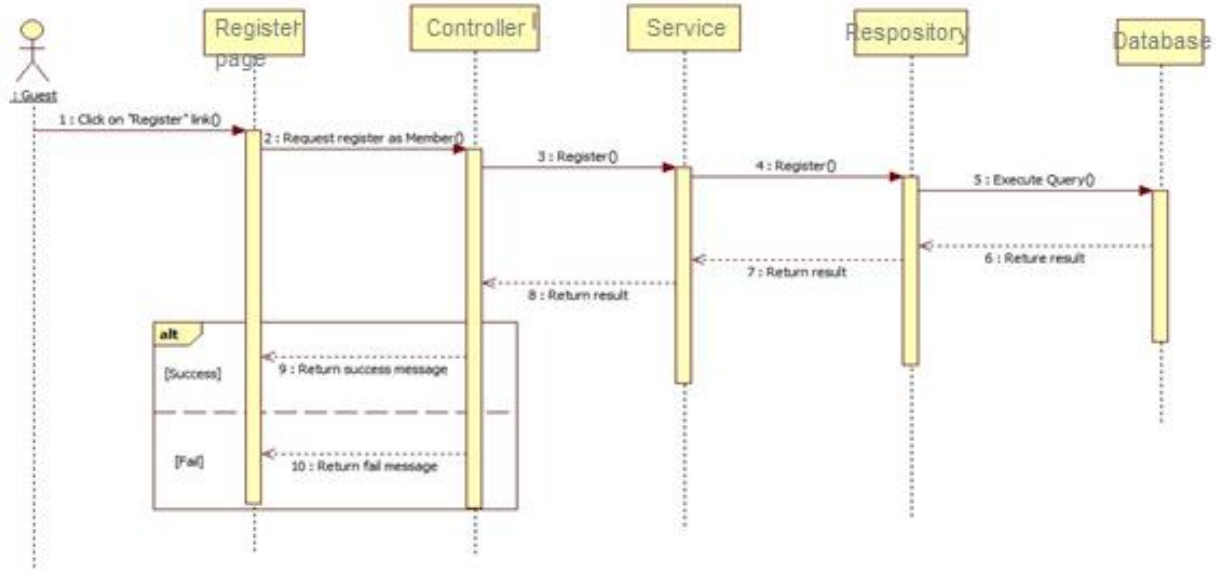
3.1.Start a new post



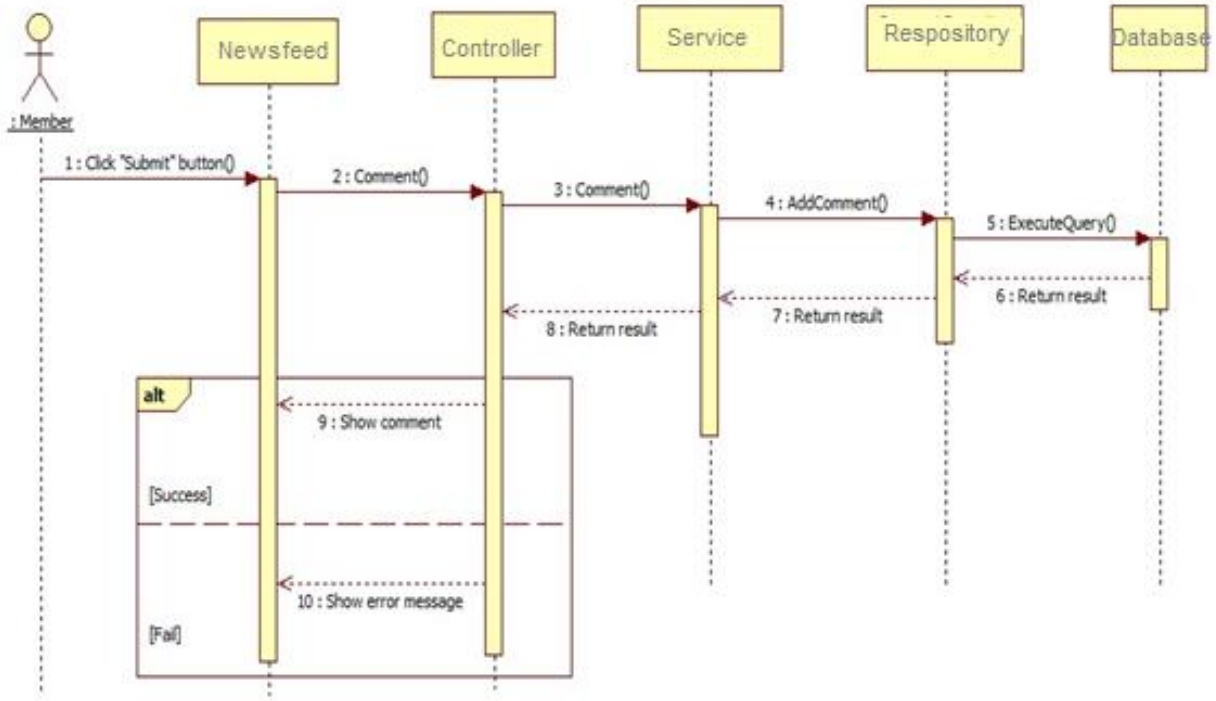
3.2.Search user



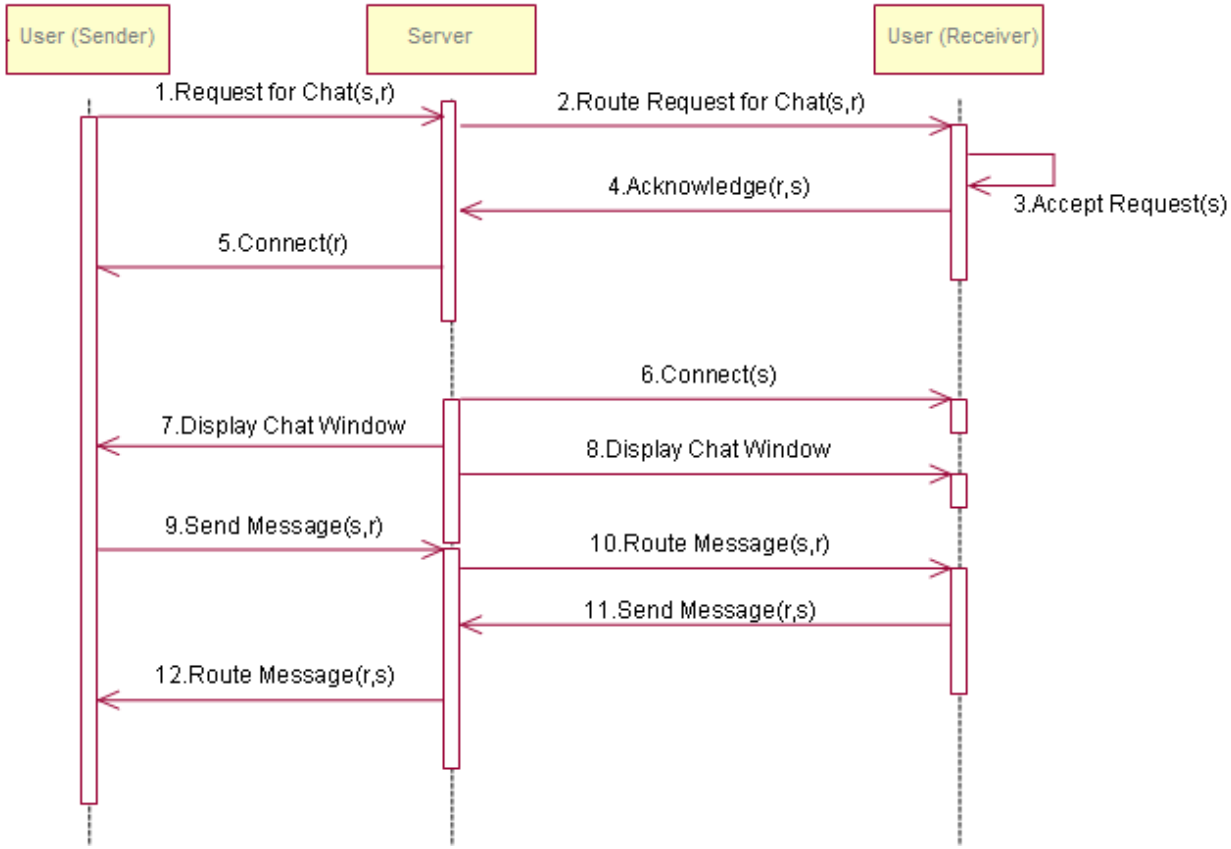
3.3.User register:



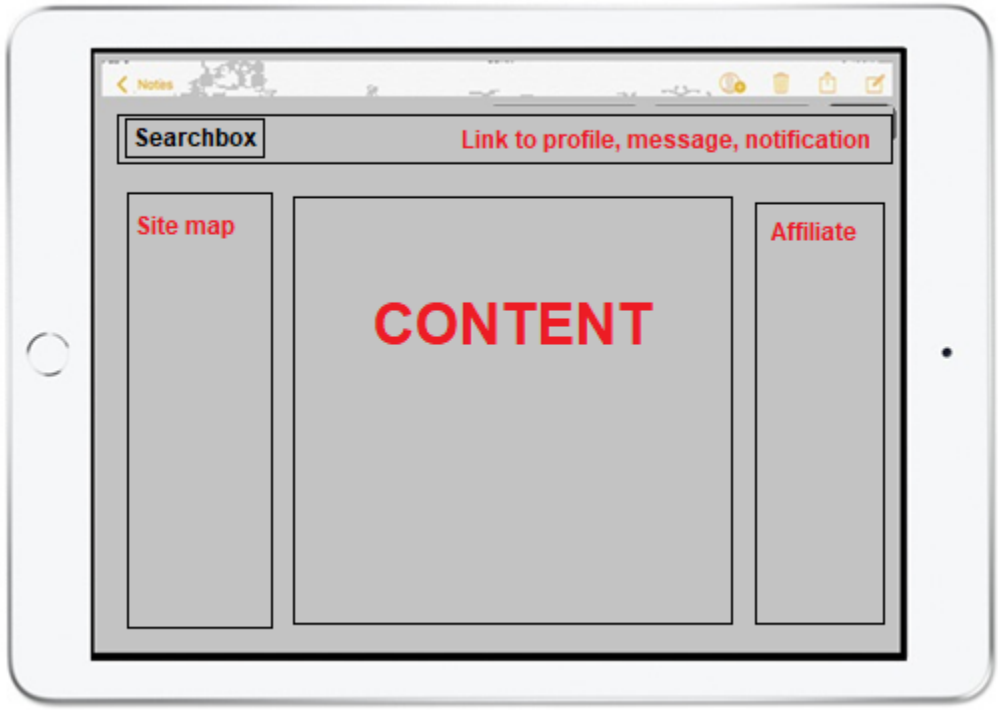
3.4.Comment to post



3.5.Sending message

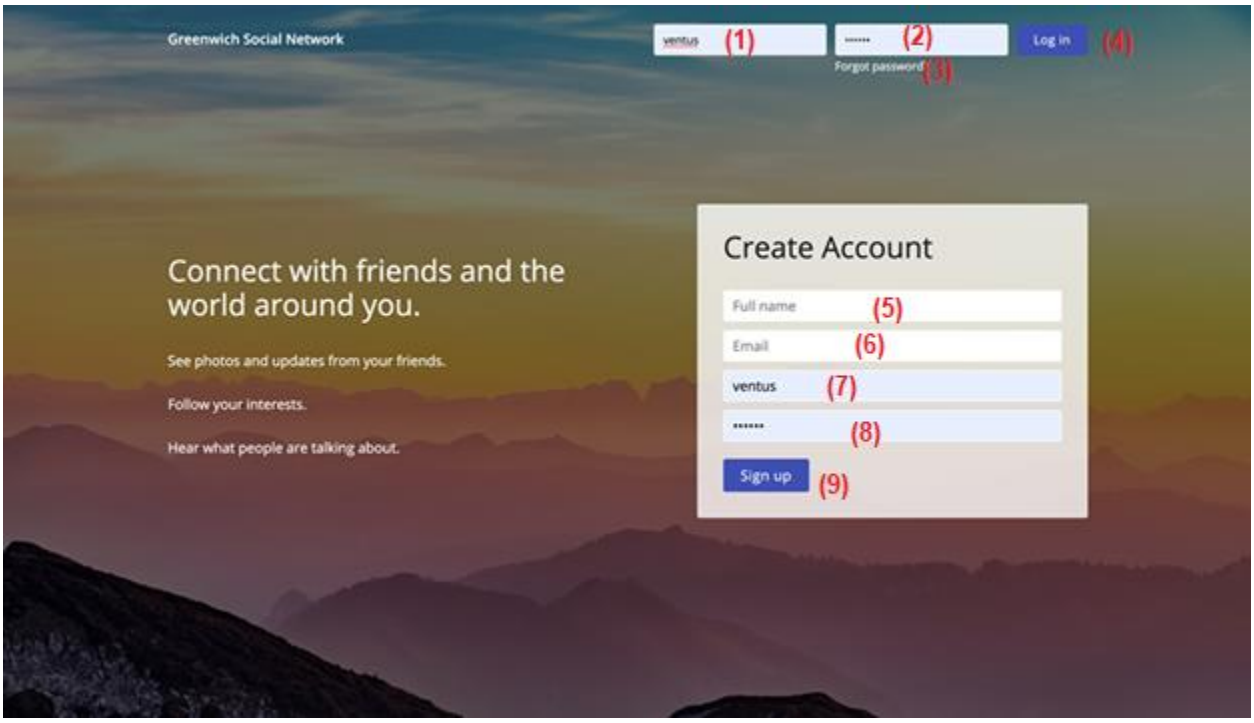


4. GUI design basic and analysis (Prototyping)



5. Features include with screenshots

5.1. Login and Registration in the same form



Fields

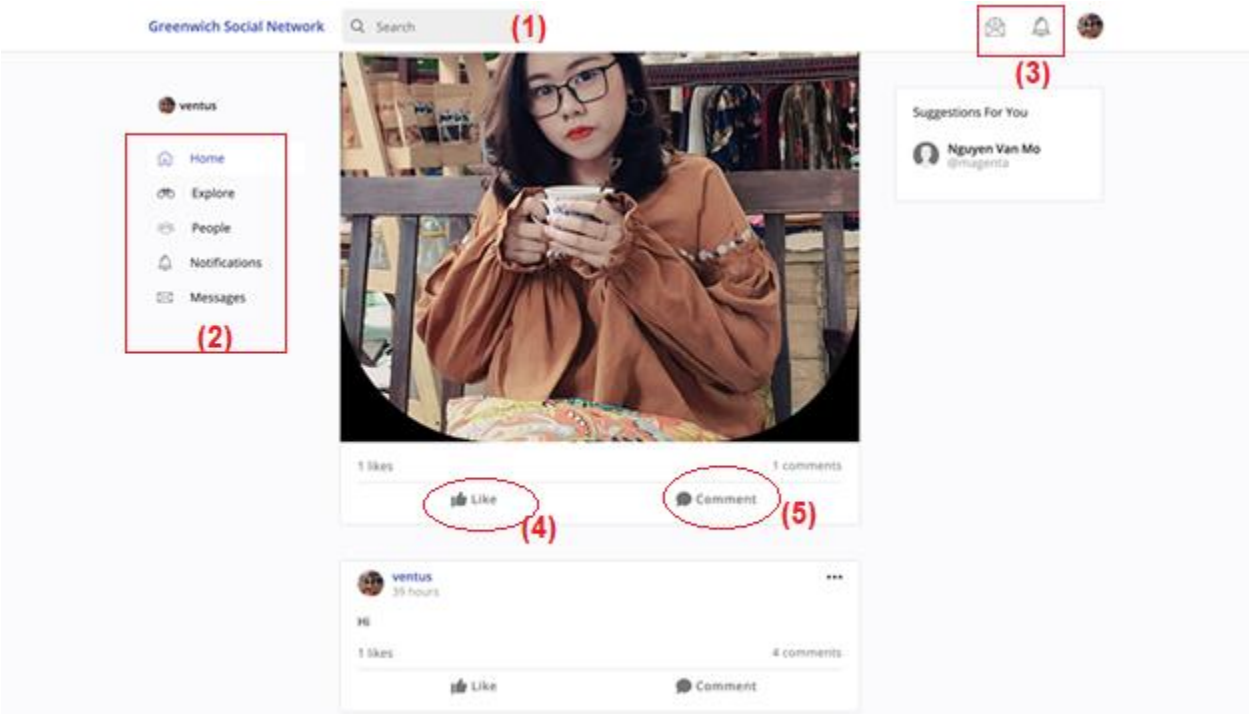
No	Field Name	Description	Read only	Mandatory	Control Type	Data Type	Length
(1), (7)	Username	Fill username	No	Yes	Textbox	String	N/A
(2),(8)	Password	Fill password	No	Yes	Textbox	String	N/A
(5)	Full name	Fill full name	No	Yes	Textbox	String	N/A

(6)	Email	Fill email	No	Yes	Textbox	String	N/A

Buttons/Hyperlinks

No	Function	Description	Validation	Outcome
(4)	Login	Click to confirm	N/A	Exit this page
(9)	Sign up	Click to confirm	N/A	Information saved & exit this page

5.2. Homepage



Fields

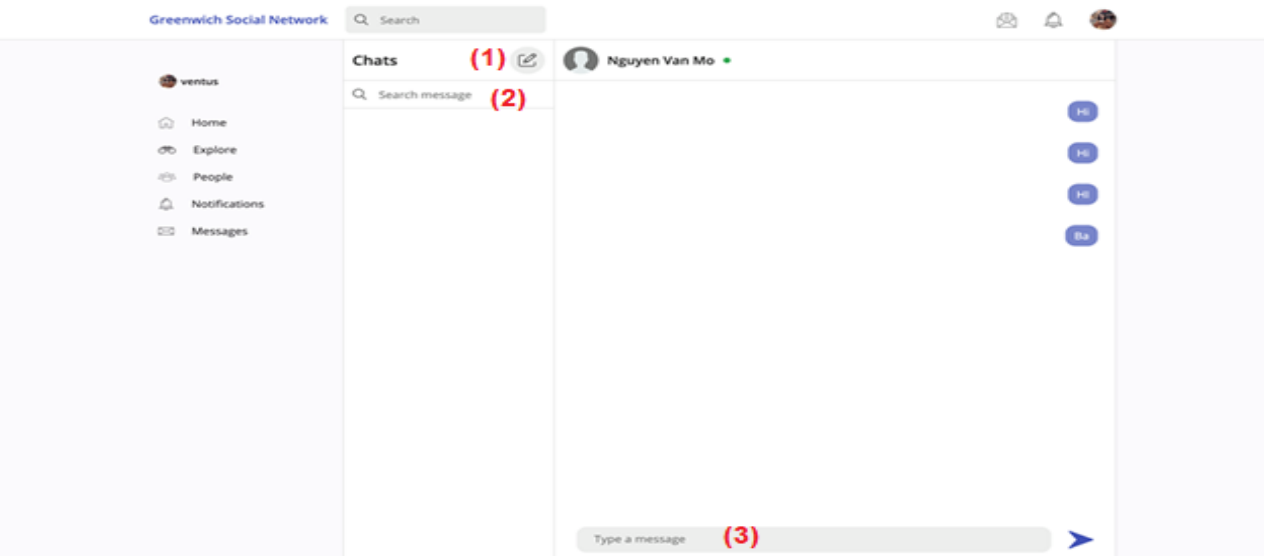
No	Field Name	Description	Read only	Mandatory	Control Type	Data Type	Length

(1)	Search	Fill account name	No	Yes	Textbox	String	N/A

Buttons/Hyperlinks

No	Function	Description	Validation	Outcome
(2)	Navigation bar list	Click to view	N/A	Transfer to new popup
(3)	Notification bar list	Click to view	N/A	Information saved & exit this page
(4)	Like label	Click to like	N/A	Like letter is in bold
(5)	Comment label	Click to give comment	N/A	Show textbox to fill

5.3. Messaging view



Fields

No	Field Name	Description	Read only	Mandatory	Control Type	Data Type	Length
(2)	Message looking up	Account name	No	Yes	Textbox	String	N/A
(3)	Message	Fill message	No	Yes	Textbox	String	N/A

Buttons/Hyperlinks

No	Function	Description	Validation	Outcome
(1)	New chat	Click to start new chat	N/A	Show chat box

6. Product Implementation

6.1. Prima migration model:

The following code describe the structure of Post, Follow and User model with Prisma schema type system that look pretty similar to GraphQL schema type. Then, we run the “npx prisma migrate save”, the corresponding SQL script would be generated, and, after “npx prisma migrate up”, those script will run and create the corresponding table in the database.

Now, if you wonder why those Prisma type look like GraphQL type? That is because after we run “npx prisma generate”, a client file is generated to our node-module. From then every data manipulation query that we created will through mini-Apollo Server that make our sever-side become an Apollo Client that query and mutation data to database-server. That is the thing that make Prisma different to every other traditional ORM, and specially in help us in this investigation about the capability of GraphQL.

```
model User {
  id                       Int            @default(autoincrement()) @id
  fullName                 String
  email                    String         @unique
  username                 String         @unique
  passwordResetToken       String?
  passwordResetTokenExpiry DateTime?
  password                 String
  image                    String?
  imagePublicId            String?
  coverImage               String?
  coverImagePublicId       String?
  isOnline                 Boolean        @default(false)
  posts                    Post[]
  likes                    Like[]
  comments                 Comment[]
  followers                Follow[]       @relation("FollowingUser")
  following                Follow[]       @relation("UserFollowing")
  notifications            Notification[]
  messages                 Message[]
  createdAt                DateTime       @default(now())
  updatedAt                DateTime       @updatedAt
  userId                   Int?
  Notification             Notification[] @relation("NotificationToAuthor")
  Message                  Message[]      @relation("MessageToSender")
}

model Post {
  id            Int            @default(autoincrement()) @id
  title         String?
  image         String?
  imagePublicId String?
  author        User           @relation(fields: [authorId], references: [id])
  authorId      Int
  likes         Like[]
  comments      Comment[]
  createdAt     DateTime       @default(now())
  updatedAt     DateTime       @updatedAt
  Notification  Notification[]
}

model Follow {
  id           Int           @default(autoincrement()) @id
  user         User          @relation("FollowingUser", fields: [userId], references: [id])
  userId       Int
  follower     User          @relation("UserFollowing", fields: [followerId], references: [id])
  followerId   Int
  createdAt    DateTime      @default(now())
  updatedAt    DateTime      @updatedAt
  Notification Notification[]
}
```

6.2. Query - Get Authenticated User Resolver:

The following code is an example of how we query data through Prisma ORM. As we can see, when the authUser is not null, we do the query and find user in database, then describe the return data just like a GraphQL client describe to server.

```
/**
 * Gets the currently logged in user
 */
getAuthUser: async (root, args, { authUser, Message, User }) => {
  if (!authUser) return null

  // If user is authenticated, update it's isOnline field to true
  let user = await prisma.user.update({
    where: {
      email: authUser.email,
    },
    data: {
      isOnline: true,
    },
    include: {
      posts: {
        orderBy: {
          createdAt: 'desc',
        },
      },
      likes: true,
      followers: {
        select: {
          id: true,
          followerId: true,
        },
      },
      following: {
        select: {
          id: true,
          userId: true,
        },
      },
      notifications: {
        where: {
          seen: false,
        },
        include: {
          author: true,
          follow: true,
          like: {
            include: {
              post: true,
            },
          },
          comment: {
            include: {
              post: true,
            },
          },
        },
      },
    },
  })
```

6.3. Query - Get User News Feed

The following code is an example of how we paginate the posts of user news feed. As we can see, we count the number of posts that user could load, then limit the number of posts that load each time (page size) then the loaded posts by a “skip” variable that start from zero (0).

```
getFollowedPosts: async (root, { userId, skip, limit }, { Post, Follow }) => {
  // Find user ids, that current user follows
  let userFollowing = []
  const follow = await prisma.follow.findMany({
    where: {
      followerId: parseInt(userId),
    },
    select: {
      userId: true,
    },
  })
  follow.map((f) => userFollowing.push(f.userId))

  const query = {
    OR: [{ authorId: { in: userFollowing } }, { authorId: parseInt(userId) }],
  }

  const followedPostsCount = prisma.post.count({where:query})
  const followedPosts = prisma.post.findMany({
    where: query,
    include: {
      author: {
        include: {
          followers: true,
          following: true,
          notifications: {
            include: {
              author: true,
              follow: true,
              like: true,
              comment: true,
            },
          },
        },
      },
      likes: {
        select: {
          id: true,
          userId: true,
          postId: true
        },
      },
      comments: {
        include: {
          author: true,
        },
        orderBy: {
          createdAt: 'desc'
        }
      },
    },
    skip: skip,
    first: limit,
    orderBy: {
      createdAt: 'desc'
    }
  })

  return { posts: followedPosts, count: followedPostsCount }
},
```

6.4.Mutation - Follow User:

The next code is a different example about how we “mutate” the data with Prisma that when we use the Prisma-client API to create new relation the that connect two users.

```
 *
 * @param {string} userId
 * @param {string} followerId
 */
createFollow: async (
  root,
  { input: { userId, followerId } },
  { Follow, User }
) => {
  const follow = await prisma.follow.create({
      data: {
          user: {
              connect: {id: parseInt(userId)}
          },
          follower: {
              connect: {id: parseInt(followerId)}
          }
      }
  })

  return follow;
},
/**
```

6.5. Create Apollo Server:

The next is one of the most important function of our Apollo Server. This return new Apollo Server that build form the input “schema” then check the authentication token attached in the HTTP header request. The, the subscription is turn on to make user receive real-time notification and message. The final step is update user status to online to notify all other users..

```
export const createApolloServer = (schema, resolvers, models) => {
  return new ApolloServer({
    typeDefs: schema,
    resolvers,
    context: async ({ req, connection }) => {
      if (connection) {
        return connection.context;
      }

      let authUser;
      if (req.headers.authorization !== 'null') {
        const user = await checkAuthorization(req.headers['authorization']);
        if (user) {
          authUser = user;
        }
      }

      return Object.assign({ authUser }, models);
    },
    subscriptions: {
      onConnect: async (connectionParams, webSocket) => {
        // Check if user is authenticated
        if (connectionParams.authorization) {
          const user = await checkAuthorization(connectionParams.authorization);

          // Publish user isOnline true
          pubSub.publish(IS_USER_ONLINE, {
            isUserOnline: {
              userId: user.id,
              isOnline: true,
            },
          });

          // Add authUser to socket's context, so we have access to it, in onDisconnect method
          return {
            authUser: user,
          };
        }
      },
```

6.6. Apollo Server Schema:

The following code is an example of how a schema is described at the server-side in order to allow to request whatever data they need. These schemas are the way that server-side and client-side communicate, in order to consist the data reliable.

```
Minh Nguyen, 12 days ago | 1 author (Minh Nguyen)
import { gql } from 'apollo-server-express';      Minh Nguyen, 12 days

/**
 * Comment schema
 */
const CommentSchema = gql`
  # -----------------------------------------------------------
  # Model Objects
  # -----------------------------------------------------------
  type Comment {
    id: ID!
    comment: String!
    author: ID
    post: ID
    createdAt: String
  }

  # -----------------------------------------------------------
  # Input Objects
  # -----------------------------------------------------------
  input CreateCommentInput {
    comment: String!
    author: ID!
    postId: ID!
  }

  input DeleteCommentInput {
    id: ID!
  }

  # -----------------------------------------------------------
  # Return Payloads
  # -----------------------------------------------------------
  type CommentPayload {
    id: ID
    comment: String
    author: UserPayload
    post: PostPayload
    createdAt: String
  }

  # -----------------------------------------------------------
  # Mutations
  # -----------------------------------------------------------
  extend type Mutation {
    # Creates a post comment
    createComment(input: CreateCommentInput!): Comment

    # Deletes a post comment
    deleteComment(input: DeleteCommentInput!): Comment
  }
`;

export default CommentSchema;
```

6.7. Apollo Client Schema:

The following is an example about the client-side create a request and dynamically describe the response data to server in order to display without neither over or under fetching. As we can see, the “userPayload” is define only because it is multiple used data, other field such ass followers or notifications are depend on the requirement of UI.

```
const userPayload = `
  id
  username
  email
  fullName
  image
  imagePublicId
  coverImage
  coverImagePublicId
  createdAt
`;

/**
 * Gets specific user by username
 */
export const GET_USER = gql`
  query($username: String, $id: ID) {
    getUser(username: $username, id: $id) {
      ${userPayload}
      isOnline
      posts {
        id
      }
      following {
        id
      }
      followers {
        id
      }
      notifications {
        id
        author {
          id
          username
        }
        follow {
          id
        }
        like {
          id
        }
        comment {
          id
        }
      }
    }
  }
`;
```

7. Testing:

7.1. Testing plan

1. Scope of The Test

1.1 The social network application

1.2.Testing tool: Observation.

2. Testing Environment

2.1 Testing app: All the functions developed

2.2 Software:

Documentation tool	Microsoft Word 2019
DBMS	Navicat Premium 12
Operating System	Windows 10, MacOS High Serria
Code Editor & Build	Visual Studio Code & Terminal

3. Test Detail

3.1 Test identification: Test UI and Test features

3.2 Test objective: All of the main functions

3.3 Test level (unit, integration or system tests)

a) Unit: Transaction, UI and Data Flow.

b) Integration: Functionally, Performance.

c) System: Error handling, Transaction.

3.4 Test case

3.4.1 Sign in case

3.6.1.1 Sign in with valid username and password.

3.6.1.2 Sign in with invalid username.

3.6.1.3 Sign in with incorrect password.

3.6.1.4 Sign in when leave username as blank.

3.6.1.5 Sign in when leave password as blank.

3.6.1 Post management case

3.6.2.1 Create new post with image.

3.6.2.2 Create new post without image.

3.6.2.3 Create post with invalid size of file image.

3.6.2.4 Delete post.

3.6.2.5 Copy link post.

3.6.2.6 Delete post with like.

3.6.2.7 Delete post with comment.

3.6.2 User interaction case

3.6.3.1 Follow user.

3.6.3.2 Unfollow user.

3.6.3.3 Search for user.

3.6.3.4 Comment user post.

3.6.3.5 Like user post.

3.6.3.6 Delete comment on user post.

3.6.3.7 Unlike user post.

3.6.3 Messaging case

3.6.4.1 View all conversation.

3.6.4.2 Search for user to send message.

3.6.4.3 Send new message.

3.6.4.4 Filter conversation.

3.6.4.5 View unread message received.

3.6.4 Notification case

3.6.5.1 View all notification.

3.6.5.2 View unread notification.

3.6.5.3 Redirect to user occur notification.

3.6.5.4 Redirect to post of like notification.

3.6.5.5 Redirect to post of comment notification.

3.6.5 Homepage action case

3.6.6.1 Load news feeds. 3.6.6.2 Redirect to Explore new post. 3.6.6.3 Redirect to People view. 3.6.6.4 Redirect to message view. 3.6.6.5 Redirect to suggested user view. 3.6.6.6 Like or comment post. 3.6.6.7 Load more news to feeds. 3.6.9.1 Create new post to feeds.
4. Test Schedule 4.1 Construction: 1 week 4.2 Examine: 1 week 4.3 Fix problems: 2 weeks

7.2. Software testing principles:

Software cycle is counted from the time of request (new or upgraded grant) until the software meets the delivery requirements. In each cycle, people conduct many stages: initiation dynamic, detailed, realistic and transfer. Each stage is often done by the mechanism of repetition to many times to achieve better results. In each iteration, we often perform multiple workflows concurrently (to make the best use of human resources): Capture requirements, analyze functions, design, implement and test to try on. After each iteration of a certain job, we have to create result (artifacts), the result of this step / job is the input data of another step / job. If the information is not good, it will severely affect đ results in subsequent steps / activities.

Some common problems in software development:

1. Calculation is incorrect, correction of wrong data.
2. Searching for wrong data required.

3. Mishandling relationship between data.
4. Performance of software is still low.
5. Unreliable software results or performance.
6. Control software security, not yet incubated.

Type of test	Test technique
Unit Testing	White Box, Black Box
Integration Testing	Black Box, White Box
Functional Testing	Black Box
System Testing	Black Box
Acceptance Testing	Black Box

8. Evaluation:

The application relatively meets the initial requirements for a social network using graphQL as a data API. The page displays content clearly and receives exactly what data to display. The messaging feature shows the real-time nature of the node express server. Furthermore, navigation between screens is smooth and user interactions also occur with relatively low latency. Notifications are sent to users almost as soon as events like follow, comment, or like are triggered.

However, the functions of the application are still relatively few, not fully showing the effectiveness of the graphQL model implemented by Apollo Server and Apollo Client. Moreover, the interface is smooth but still rudimentary and monotonous, needs more improvement to increase the experience of excitement for users.

Chapter VII: Conclusions

For this project, I have learned about, firstly, the graphQL query language and how it could be turned into a full-stack data transfer achitecture. Moreovers, I have concluded for myself the valuable experience developing a social network. Such as, the important of evaluation and deciding the database model to implement that suitable for the data structure in a social network. Additionally, I have accumulated for myself the lessons about should a social network system work, and the important of real-time data transfer subcription in social media application services. About the result of this project, I have create a full-stack social network service that present the effectiveness in handling complex and dynamic data request that the traditional RESTful API would hard to achieve. I have also successfully implement the new generation of database ORM like Prisma to handle not only the database connection but also data migration and manipualtion. Last but not least, I have present all of those back-end product on a well-designed UI of social network that include the most basic function of social network such as Post, Comment, Like, News Feed or Message.

- **Futurre work:**

Back-end: Currently, the product can only work on one database at a time: MongoDB or MySQL. Howevers, data in social network is complex and dynamic, some solution of using a mixing of two database such as: SQL database for storing complicated data user relations and activity streams and NoSQL database for portable and non-relation data such as notifications, messages or user interactions.

Front-end: Some of the interesting feature of social network should be considered to implement. The user interface should be modify to be more attractive to users.

References

(1) Boyd, d., & Ellison, N. (2007). Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication*, 13(1), 210-230.

(2) Hawkins, Tim; Plugge, Eelco; Membrey, Peter (September 26, 2010), The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing (1st ed.), Apress, p. 350

(3) Mongoose v5.9.13 Getting Start, Licensed under MIT. Copyright 2011 \

< mongoosejs.com/docs>

(4) Introduction to GraphQL, Copyright © 2020 The GraphQL Foundation

<https://graphql.org/>

(5) Prisma Documentation, Prisma © 2018 – 2020 <https://www.prisma.io/docs/>

(6) Morpheus Data, MySQL vs. MongoDB: The Pros and Cons When Building a Social Network

(April 1, 2015) <https://morpheusdata.com/blog/2015-04-01-mysql-vs-mongodb-the-pros-and-cons-when-building-a-social-network>

(7) Nick Karnik, Introduction to Mongoose for MongoDB (Feb 20, 2018),

<https://www.codementor.io/@theoutlander/introduction-to-mongoose-for-mongodb-gw9xw34el>

(8) Introduction to JSON Web Tokens, <https://jwt.io/introduction/>

(9) React A JavaScript library for building user interfaces, Copyright © 2020 Facebook Inc.,

<https://reactjs.org/>

Project Proposal

Title:

AN INVESTIGATION OF GRAPHQL

INTO

DEVELOPING SOCIAL NETWORK

Researcher's name:

Nguyễn Hoàng Minh

Supervisor's name:

Lâm Nguyễn Trung Nam

Programme:

Top-Up SE HCM

Department:

Computing

Institution:

University of Greenwich, Ho Chi Minh Campus

Date:

February 18th, 2020

Introduction

Background

In 2015, Facebook (that big social network company that also gave us React) released a new open-source data query language (QL) named GraphQL. GraphQL became so popular that by the end of 2018 it detached from Facebook into its own foundation that is maintained by the Linux Foundation — a non-profit technology consortium founded in 2000.

But before GraphQL, there was REST, which stands for Representational State Transfer. REST is a standard that defines a set of rules that developers needed to follow while creating any kind of web services. Web services that are based on these rules were termed to be RESTful. The reason why developers followed these rules is that it provided a kind of interoperability between different systems.

But why GraphQL? How does it differ from REST? And should become better effectivness in developing social network application?

Problem Statement

GraphQL follows the same set of constraints as REST APIs, but it organizes data into a graph using one interface. Objects are represented by nodes (defined using the GraphQL schema), and the relationship between nodes is represented by edges in the graph. Each object is then backed by a resolver that accesses the server's data.

Howevers, what are the core difference between REST APIs and GraphQL? Where GraphQL has the upper hand over REST APIs?

The most probably answer should be social network application development.

Before Facebook released it to the people as open-source, GraphQL was internally used by Facebook since 2012 as an alternative to the REST style of architecture.

GraphQL gives us the ability to ask for specific data. This gives the client-side systems a greater amount of control over what information to request and receive. This way, the network usage rate gets reduced like anything!

Aim

The final aim of this project should be the capability in applicating GraphQL into a fullstack model of a social network that cover from back-end to front-end, espacially the effectiveness of data transaction process.

Moreovers, an evaluation of what improvement and how do it occur when GraphQL is implemented, compared to others data API model, especially the tradiational and famous REST API.

Objectives

Project Report

The capability of delivering a final report that cover the overall process from researching, evaluating and identifying solutions to implementing, testing and completing the product.

Particularly, the product report has the following sub-objectives:

Researching about the GraphQL query language and how should it be implemented into a data API.

Read the GraphQL query language offical documents.

Synthecing article, journal and tutorial about GraphQL and how to implement it.

Research about data API ? How many type of data API and how GraphQL beat out them in developing social network?

Point out the problems of GraphQL and the difficult should be considered in implementing the model.

Should GraphQL be mixed with traditional Restful?

Researching about the relavent frameworks, tools and library that should be considered to used in implementation. Evaluating them, point out the good, the bad and the ugly and how do they beat out others to be come in used.

Find out the most suitable and handlable with current skills and ability that could be used to create the back-end data API. (considered NodeJS).

Research about the database types (NoSQL, SQL or mixing) and database languages (postgre, MSQLS, mongoDB, etc) that should be used in this project? Is it neccessary to mix between two of more? (considered mixing between MongoDB for NoSQL and postgre for SQL).

Considering GraphQL data server, ORM and middleware in order to simplize the developing process...

Choosing the front-end development language and framwork to create the social network application (considered React-native for the mobile version and eactjs for the web version if capable).

Noted down daily research and implementtation to synthetic the process.

Daily report should be considered to be made in order to guarantee all the process is noted down and evaluated.

Creating list of requirements, products backlog and development test case in order to process the final evaluation of delivered model.

Create list of requirement for the product.

Create product backlog and weekly task that should be done for completing project before deadline.

Create test case and test result evalution for majority function of final model.

Syntheticing created ERD diagrams for database models, XML structures for products functional and UI mockup for later report.

Product Design

The capability of delivering the conceptual design and functional diagrams from database to back-end data API and front-end applcations, which should be considered to include following sub-objectives:

The ERD diagrams of database models.

The XML Schema for back-end data API using graphQL.

The front-end social network application UI design.

The specfic model design for implementing GraphQL.

Final Product

The capability of delivering the final product, which should include the following sub-objectives:

A data API with fully function implemented with GraphQL.

Data API should show of the effective of GraphQL models in data transaction process.

A social network application with capability of connecting to GraphQL data API effectively.

Application should have acceptable UI/UX desgin and follow exactly the principle requirement has been defined.

An evaluation report and Statistical data about the effectiveness of graphQL in developing social networking applications.

Evaluation and Reflection

The fully project cover evalutation and reflection about both the product development process and research that should include the following sub-objectives:

Product evaluation via testing process according to designed test case.

Research report evaluation based on the completement of report.

Pointed out critical reflection about the working process about what have been done? How it have been done? Which should be done better?

Research design and methods

The primary research method for this study is literature review and conceptual modeling. Constraint identification and classification through a structured approach is the very first step toward a “zero-constraint” environment. This study will first review various types of constraints in construction and their characteristics. Based on this understanding, a classification method will be developed to categorize constraint factors for the purpose of constraint identification and modeling.

In the second stage of this study, existing constraint modeling methods will be identified based on a comprehensive review of current industry practices and academic researches. Finally, once the constraint classification and modeling techniques are identified, a conceptual framework for total constraint management will be outlined.