Index

Experiment 1

Introduction to OOPs and OOAD

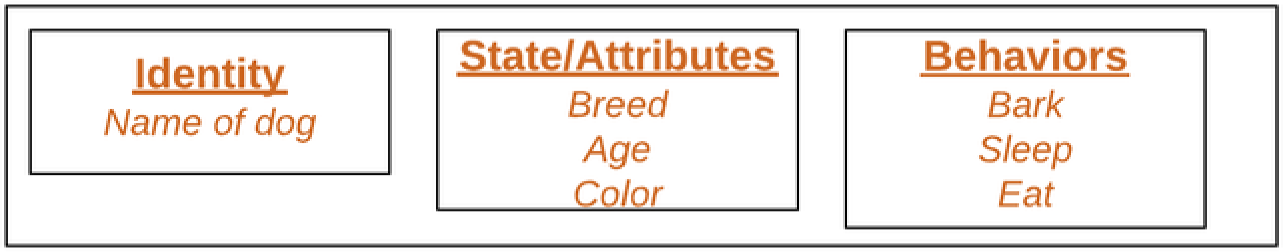
OOPs: Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

OOPs Concepts:

- Class
- Objects
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

1. Class:
A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.
For Example: Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, Car is the class, and wheels, speed limits, mileage are their properties.

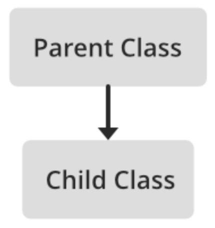
2. Object:
It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior. Each object contains data and code to manipulate the data.
For example, “Dog” is a real-life Object, which has some characteristics like color, Breed, Bark, Sleep, and Eats.



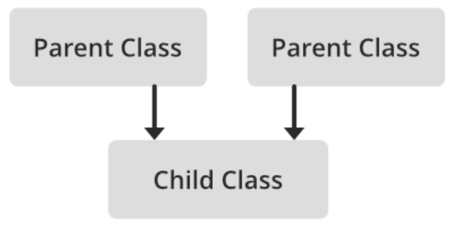
3. Data Abstraction:
Data abstraction is one of the most essential and important features of object-oriented programming. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation. Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

4. Encapsulation:
Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.

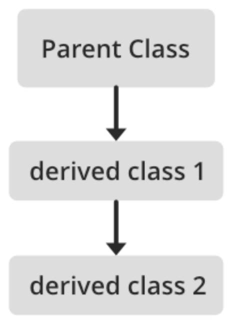
5. Inheritance:
Inheritance is an important pillar of OOP(Object-Oriented Programming). The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.



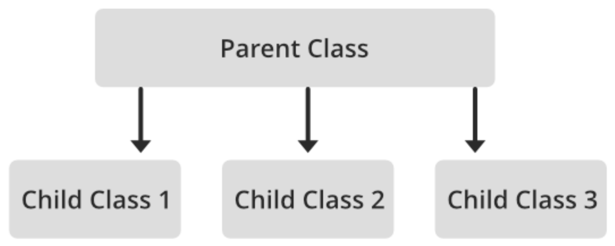
Single inheritance



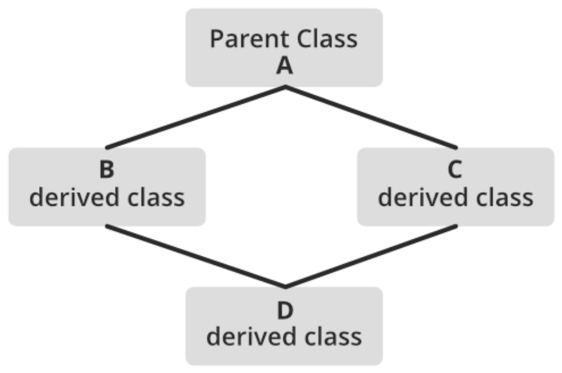
Multiple inheritance



Multilevel inheritance



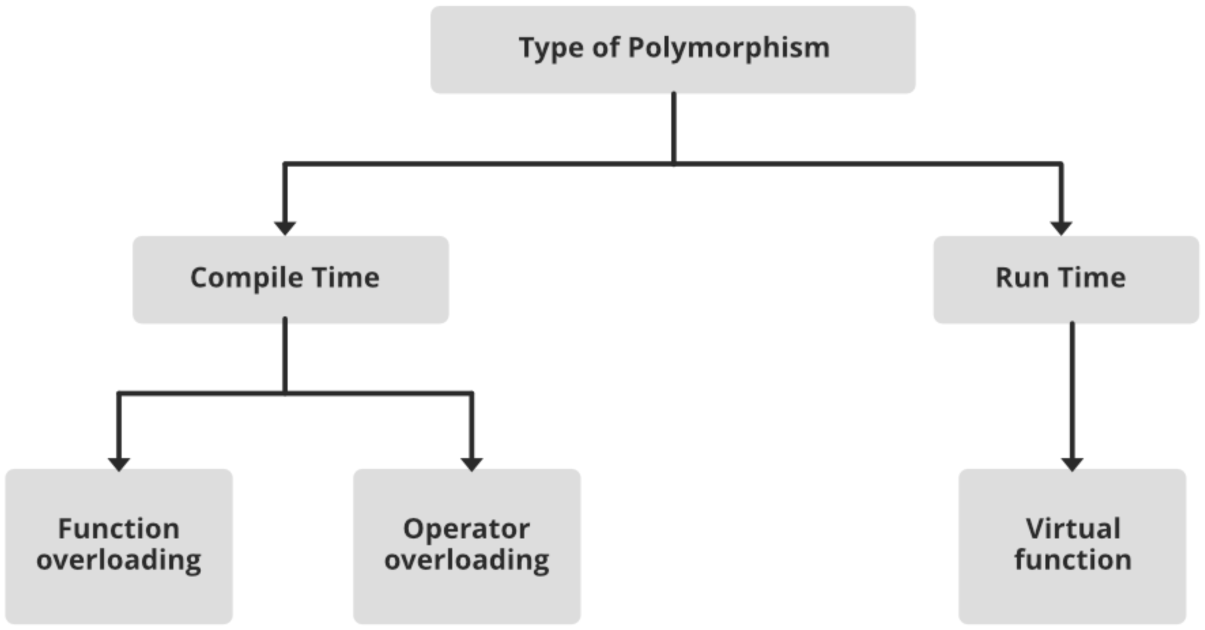
Hierarchical inheritance



Hybrid Class

6. Polymorphism:

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.



7. Dynamic Binding:
In dynamic binding, the code to be executed in response to the function call is decided at runtime. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. Dynamic Method Binding One of the main advantages of inheritance is that some derived class D has all the members of its base class B. Once D is not hiding any of the public members of B, then an object of D can represent B in any context where a B could be used. This feature is known as subtype polymorphism.

8. Message Passing:
It is a form of communication used in object-oriented programming as well as parallel programming. Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure, and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function, and the information to be sent.

OOAD - *Object-Oriented Analysis and Design* (OOAD) is a software development approach that models a system as a group of objects. Each object represents some real world entity within the system being modeled, and has its own *attributes* and *operations*. A range of models can be created using such objects to reflect the structure and behaviour of the system. One of the most widely used notations for depicting objects and the way they interact with each other is the *Unified Modeling Language* (UML). UML was adopted by the *Object Management Group* (OMG) in 1997 as the standard for object-oriented modelling, and combines widely accepted concepts from a number of object-oriented modelling techniques.

- **Objects and classes**
  Essentially, an object is an abstract representation of some real-world entity, such as a person or a bank account. It will hold data items (attributes) about the entity (for example, name or account number). It will define operations (or methods) to report the values held by attributes, or modify them in some way. The operations defined for a class provide it with an interface through which it can communicate with other objects. The object will be identified by a unique name. A class is a template that defines the attributes and operations that are common to the objects created from it. Looking at this from a different perspective, an object is an instance of a class. These ideas are common to both the object modelling notations used to analyse and design systems, and the object-oriented programming languages used to implement those systems.

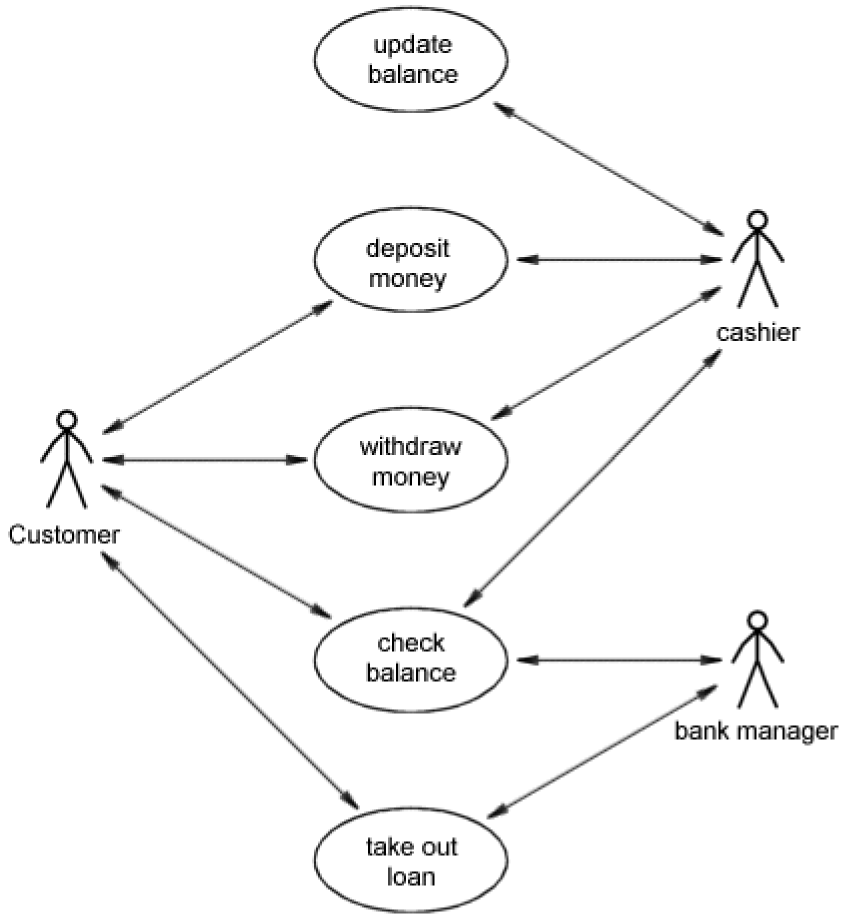
- **Modelling the system structure**
  Modelling languages like UML provide a way to create a model of a system that can be understood by both system developers and users, and can form the basis for an ongoing dialogue between these two groups of people. Everything within the system of interest is described in terms of objects, the properties of those objects, how objects can act and be acted upon, and the relationships that exist between objects. The models themselves consist primarily of diagrams, and the various types of diagram used employ a consistent notation throughout to maintain clarity. The aspects of a system that can be represented diagrammatically include its structure, which for the most part is relatively static, and its behaviour, which is dynamic and often complex.
  Other types of UML diagram used to represent the structure of the system include:

1. **Component diagrams** - a component is a collection of closely related classes that provide a specific set of services. A component diagram shows the structure of (usually) a single component within the overall system architecture. Each component diagram depicts the detailed class structure of a component, including the class attributes and operations, and the dependencies and relationships between classes.
2. **Package diagrams** - a package diagram shows the hierarchical structure of the system design. It illustrates how related system elements are grouped, and the dependencies and relationships between the various groupings. Package diagrams can be used to break a large and complex design down into a number of smaller and more easily managed designs.
3. **Deployment diagrams -** the deployment diagram shows the physical architecture of the system in terms of its implementation in hardware, how the system components are assigned to the various nodes, and how the nodes communicate with each other and with other hardware devices.

- **Modelling the system's behaviour**
  The behaviour of the system is modelled dynamically. You can think of it in terms of

creating a series of stories, each of which describes some aspect of how objects behave and how they interact with one another. A number of diagramming techniques are provided by UML to model the events that occur within the system, what actors are responsible for instigating them, and the actions that they trigger. We are interested in the interactions between objects, the transitions that occur, and the sequencing of events. Modelling these dynamic characteristics can often uncover additional system requirements, and result in further refinement of the system's logical or physical architecture models.

- **Use Case diagrams**
  Use-case diagrams model the system's functions from the perspective of various actors. An actor is an external entity that makes use of the system in one or more ways. The actor is usually a person, but may be an organization or another computer system. Actors may also (though not necessarily) be represented within the system by a class. The use-case models how an actor uses the system in one particular way, and the functionality provided by the system for that situation.



- **Activity diagrams**
  An activity diagram is a bit like a flowchart for a computer program, but in this case it represents activities within a system, and the events that cause objects to be in a

particular state. The following simple activity diagram illustrates what might be involved in a student attending college for a class.

The first activity is to get dressed. A decision then has to be made, depending on how much time is available before the class begins and the times of local buses. If there is enough time to get to college by bus, take the bus. If not, take a taxi. The last activity is to actually attend the class, after which the activity diagram terminates. The diagram commences at the initial node (drawn as a solid black circle), and ends at the terminating node (drawn as a bull's eye). Activities are drawn as rounded rectangles containing a brief description of the activity. A decision node is shown as a diamond, with arrows emerging from it to represent the alternative control flows.

Experiment 2

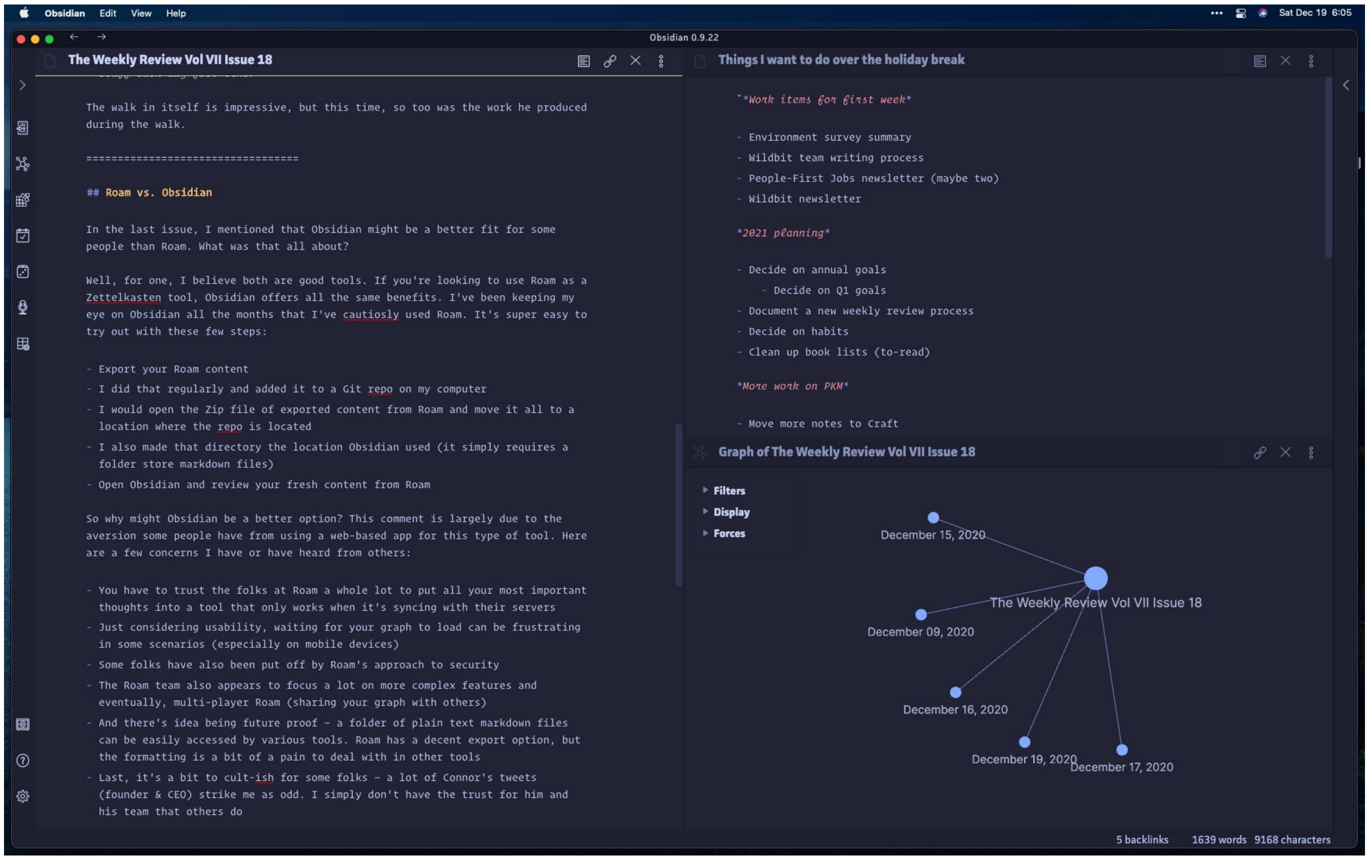
Identify the problem statement and suggested solution for the system of relevance

Notes are the important part of learning. Years ago, we make notes using **COPY-PEN**, but in that process we face many problems like bad-handwriting, damaged by external environment, can't access anywhere, connection between information is missing and many more...

There are various **NOTE-TAKING** applications available in the market like - Microsoft 365, G-edit, Notion, EverNote, Google Keep, etc. Some applications are way-too-expensive, and not build for all the platforms. Some have funky animation which slows down the notes making process. Applications like Google Keep and Notion has some privacy breaches, that makes them unsafe in terms of privacy, and they are not super extensible. In these sense, Obsidian comes into the market.

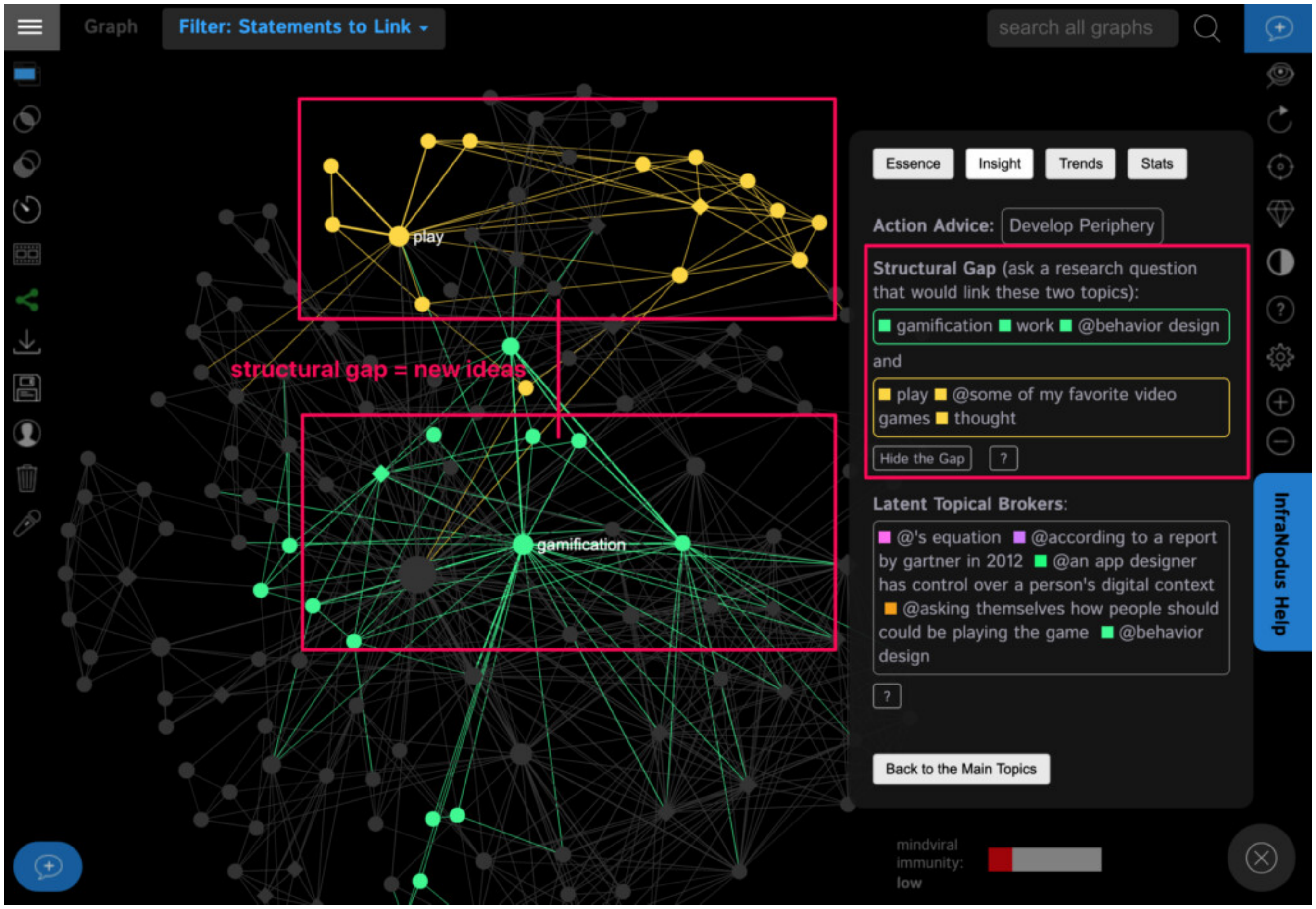
OBSIDIAN: A second brain, for you, forever.

Obsidian is a powerful **knowledge base** on top of a **local folder** of plain text Markdown files. Markdown based multi-platform note-taking application free for all. It helps to connect information together, with the help of some tags. Suggest better words for your sentences and can predict next line with the help of plugins. All notes are in **Markdown** format, so you can easily export them in any format you want. Having a huge variety of useful plugin and themes. Live Preview feature for Markdown takes it to the next level. You can publish your notes/ documentation directly to the internet using this. Obsidian cares about your privacy. Account creation in not required until you purchase any plan. Version history is also supported using some extensions. Obsidian community has huge numbers of active helping members which are from different-different region and helps to resolve your problem.


Obsidian Edit View Help
Sat Dec 19 6:05
Obsidian 0.9.22
The Weekly Review Vol VII Issue 18
Things I want to do over the holiday break
Graph of The Weekly Review Vol VII Issue 18
Filters
Display
Forces
December 15, 2020
The Weekly Review Vol VII Issue 18
December 09, 2020
December 16, 2020
December 19, 2020
December 17, 2020
5 backlinks
1639 words 9168 characters

```
The walk in itself is impressive, but this time, so too was the work he produced
during the walk.

==================================

## Roam vs. Obsidian

In the last issue, I mentioned that Obsidian might be a better fit for some
people than Roam. What was that all about?

Well, for one, I believe both are good tools. If you're looking to use Roam as a
Zettelkasten tool, Obsidian offers all the same benefits. I've been keeping my
eye on Obsidian all the months that I've cautiosly used Roam. It's super easy to
try out with these few steps:

- Export your Roam content
- I did that regularly and added it to a Git repo on my computer
- I would open the Zip file of exported content from Roam and move it all to a
  location where the repo is located
- I also made that directory the location Obsidian used (it simply requires a
  folder store markdown files)
- Open Obsidian and review your fresh content from Roam

So why might Obsidian be a better option? This comment is largely due to the
aversion some people have from using a web-based app for this type of tool. Here
are a few concerns I have or have heard from others:

- You have to trust the folks at Roam a whole lot to put all your most important
  thoughts into a tool that only works when it's syncing with their servers
- Just considering usability, waiting for your graph to load can be frustrating
  in some scenarios (especially on mobile devices)
- Some folks have also been put off by Roam's approach to security
- The Roam team also appears to focus a lot on more complex features and
  eventually, multi-player Roam (sharing your graph with others)
- And there's idea being future proof - a folder of plain text markdown files
  can be easily accessed by various tools. Roam has a decent export option, but
  the formatting is a bit of a pain to deal with in other tools
- Last, it's a bit to cult-ish for some folks - a lot of Connor's tweets
  (founder & CEO) strike me as odd. I simply don't have the trust for him and
  his team that others do
```

```
*Work items for first week*

- Environment survey summary
- Wildbit team writing process
- People-First Jobs newsletter (maybe two)
- Wildbit newsletter

*2021 planning*

- Decide on annual goals
   - Decide on Q1 goals
- Document a new weekly review process
- Decide on habits
- Clean up book lists (to-read)

*More work on PKM*

- Move more notes to Craft
```



Essence | Insight | Trends | Stats

Action Advice: Develop Periphery

Structural Gap (ask a research question that would link these two topics):

🟩 gamification 🟩 work 🟩 @behavior design

and

🟨 play 🟨 @some of my favorite video games 🟨 thought

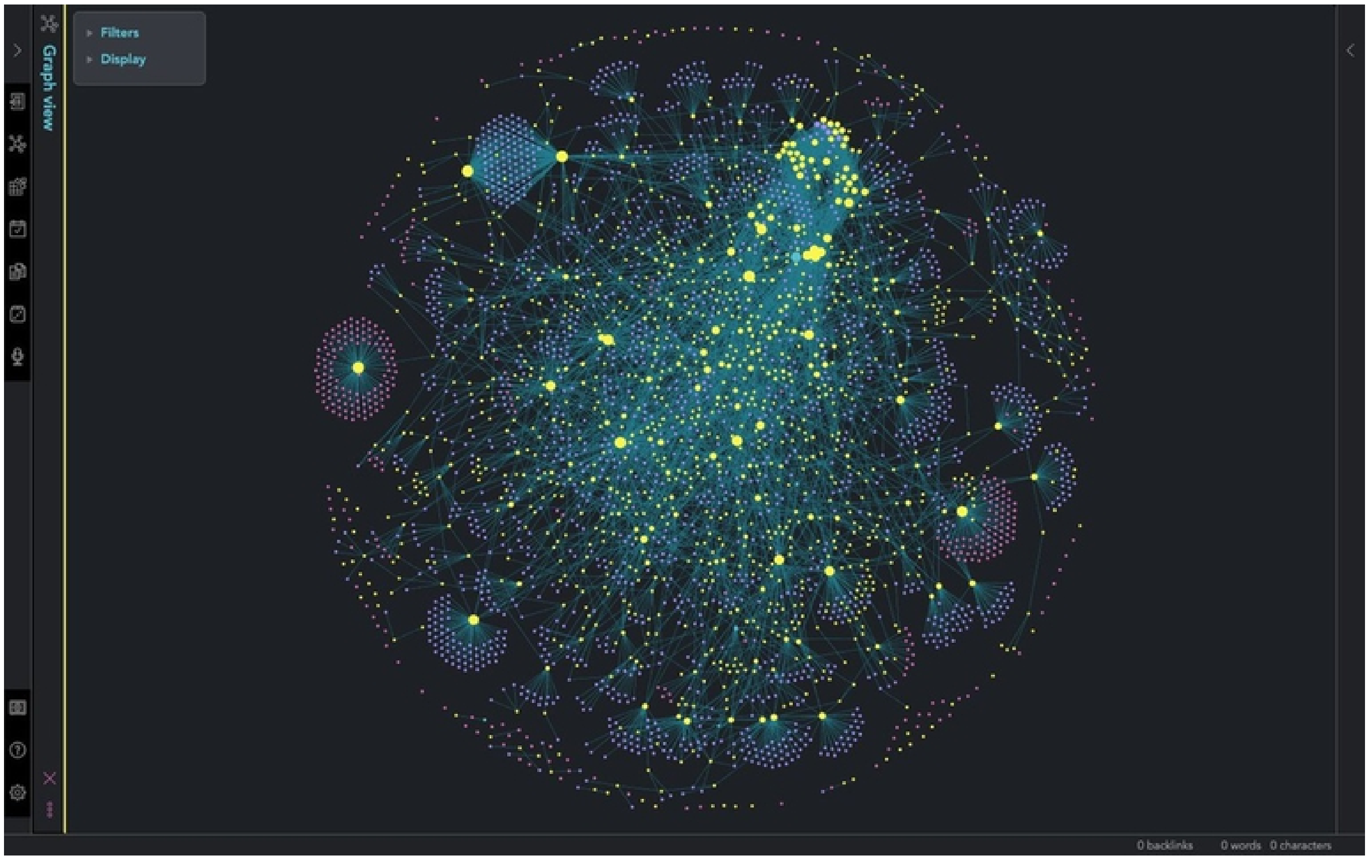
Hide the Gap ?

Latent Topical Brokers:

🟪 @'s equation 🟪 @according to a report by gartner in 2012 🟩 @an app designer has control over a person's digital context 🟧 @asking themselves how people should could be playing the game 🟩 @behavior design

?

Back to the Main Topics


Graph view
Filters
Display
0 backlinks
0 words
0 characters

Experiment 3

Identify the functional and non-functional requirements of the identified problem

Functional Requirement: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Non-Functional Requirement: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to the other. They are also called non-behavioral requirements.

OBSIDIAN's Functional Requirements

- A huge variety of themes and plugins are available.
- Highly extensible markdown note-taking app.
- Graph View for better connection of learning.
- Link and Back-Links for converging thoughts.
- Supports extended markdown (built-in).
- Store notes on local storage.
- Vibrant Community.

OBSIDIAN's Non-Functional Requirements

- New-version in every two months.
- Does not require internet connection.
- Developed majorly in JavaScript and TypeScript.
- CSS can be edited easily to personify your interface.
- Light-weight application which is good for all type of devices.
- Open Source project which continuously improving with time.
- Files are highly portable and can convert into different file formats.
- Unlimited number of knowledge bases can be created by using different Vaults.
- Directly publish your notes/documentation on web using **Publish Subscription**.

Experiment 4

Prepare a Software Requirement Specification document for suggested system

1. Introduction

```
1.1 Purpose
1.2 Scope
1.3 Definitions, Acronyms & Abbreviations
1.4 References
1.5 Overview
```

2. Overall Description

```
2.1 Product Perspective
2.2 Product Functions
2.3 User Characteristics
2.4 Assumptions & Dependencies
2.5 Apportioning of requirement
```

3. Specific Requirement

```
3.1 External Interface Requirement
        3.1.1 User Interface
        3.1.2 Functional Requirement
3.2 Software System Quality Attribute
        3.2.1 Availability
        3.2.2 Reliability
        3.2.3 Security
```

4. Supporting Information

1. Introduction

```
1.1 Purpose
```

We believe in plain text for something as important as your knowledge base. You don't want to put your own brain over someone else's neck, do you? Treat your second brain the same way. Sync is only a utility to facilitate working on multiple devices, the data will always primarily live on your hard disk. When the file system replaces the cloud, you get flexible options to work with your files: you can back them up with Dropbox, use Git to do versioning, or encrypt your disk for security. Whatever works on your file system will work on your Obsidian knowledge base. *Links are first-class citizens*. Links and connections are crucial to discovering the relations between what we know. Obsidian greatly values this, and encourages it with **Internal link** and **Graph view**, among other things.

```
1.2 Scope
```

OBSIDIAN is a basic application based on **Markdown**. It has few or no scope bloat and have 400+ community plugins and 100+ themes which can be customized with the help of **CSS**. It supports `extended markdown`, `HTML` and `CSS`. Not only that, but it stores files on your local storage, hence chances of data breach is very low. Furthermore, it doesn't have funky animation, which makes it super extensible and lite. It is available on all the major platforms: Linux, MacOS, Windows, Android and IOS.

```
1.3 Definitions, Acronyms & Abbreviations
```

Obsidian is a tool that can be used in many ways, from a simple list of notes to a very powerful knowledge management system. We suggest you start at your own pace, and build it into the tool you need.

```
1.4 References
```

Have questions?
Then you should visit our **community!**. We have active Discord and Forums, and the community is generally quite helpful.

```
1.5 Overview
```

The Purpose of this is to provide details about note-taking application: **Obsidian**. This is a markdown based note-taking application with **EXTENDED MARKDOWN** support. Store files on your local device using **VAULT**.

2. Overall Description

`2.1 Product Perspective`

The product is supposed to be an open source, under the GNU General Public License. It is a web based system implementing client-server model. The following are the main features that are included:

- Cross platform support: Offers operating support for most of the known and commercial operating systems.
- User account: The system allows the user to create their accounts in the system and provide features of updating.
- Search: search is simply local search engine based on key-words **Command Palette**.
- Discussion Forum: Provides users with a platform to discuss and help each other with their problems.

Ticketing system: Allows user to submit his issue to the **admin** in case his issues are not solved by FAQs and discussion forums.

`2.2 Product Functions`

Obsidian is a **note-taking and knowledge management app** by the people who created Dynalist, a popular online outliner. Think of it as “an IDE for your notes.” It lets you turn a collection of plain text files into a rich network of linked thought.

`2.3 User Characteristics`

It is considered that the user do have the basic knowledge of operating the internet and to have access to it. The administrator is expected to be familiar with the interface of the tech support system.

`2.4 Assumptions & Dependencies`

Binary releases for x86-64 processors are provided for Windows, Linux and Mac operating systems on a best-effort basis. They are built with GitHub runners as part of the release workflow defined in `.github/workflows/release.yml`.

The resulting binaries can be downloaded from **https://github.com/zoni/obsidian-export/releases**.

`2.5 Apportioning of requirement`

Windows

- Windows 7 and later are supported, older operating systems are not supported (and do not work).
- Both x86 and amd64 (x64) binaries are provided for Windows. Please note, the ARM version of Windows is not supported for now.

Mac

- Only 64bit binaries are provided for macOS, and the minimum macOS version supported is macOS 10.10 (Yosemite).

Linux

- The prebuilt ia32(i686) and x64(amd64) binaries of Electron are built on Ubuntu 12.04, the arm binary is built against ARM v7 with hard-float ABI and NEON for Debian Wheezy.

Whether the prebuilt binary can run on a distribution depends on whether the distribution includes the libraries that Electron is linked to on the building platform, so only Ubuntu 12.04 is guaranteed to work, but following platforms are also verified to be able to run the prebuilt binaries of Electron:

- Ubuntu 12.04 and later
- Fedora 21
- Debian 8

Hardware

About RAM and CPU, there are no information about that in Electron's docs, but Electron is based on Chromium, so it should need nearly the same requirements:

Windows

- An Intel Pentium 4 processor or later that's SSE2 capable
- 512 MB of RAM

Mac

- An Intel processor that's 64-bit
- 512 MB of RAM

Linux

- An Intel Pentium 4 processor or later that's SSE2 capable

3. Specific Requirement

```
3.1 External Interface Requirement
```

Depending on your goals, Obsidian provides two different views you can choose from when working with Obsidian:

- Select *Service Provider* if you use Obsidian for selling web hosting services.

Select *Power User* if you use Obsidian for you own needs, for example, to manage hosting on a VPS.

```
        3.1.1 User Interface
```

Designing a more minimal and intuitive interface, meaning hiding buttons and panes and showing them only when the user has an intent, e.g. when hovering. Also, a more intuitive behavior for easier navigation between notes. Make some design choices as defaults. The plugin system is a clever move, but making everything plugin-izable will make the new user experience a mess. For that, devs make design decisions as a default, either through their intuitive sense or through the most common use case.

```
        3.1.2 Functional Requirement
```

- **Portability:** CommonMark and GitHub Flavored Markdown (GFM) are supported for portability, they include syntax highlighting, markdown table and more. For additional expressiveness, Obsidian supports also tags, footnotes, internal links and more.

- **Backlinks and graph view:** to easily navigate your notes. Obsidian works like a real brain so connections between notes are highly encouraged and easily done by simply typing `[[` to get the link auto-completed.
- **Note Multiplexer:** panes can be split and resized as you need, allowing you to easily cross-reference multiple notes. Most suitable for big screens and short notes, panes can also be pinned or combined, according to your needs.
- **Plugins:** to create your personal note-taking toolkit. Plugins include page preview, daily note, tag pane, slides, random note, audio recorder, and so much more.

Themes: Obsidian allows custom CSS to help you customize the look of the app based on your preferences. You can find a list of available Community Themes **here**.

```
3.2 Software System Quality Attribute
```

The human brain is non-linear: we jump from idea to idea, all the time. Your second brain should work the same.

```
3.2.1 Availability
```

Available for **Windows**, **macOS**, **Linux (Snap)**, and **Linux (Flatpak)**. **For More platforms**.

```
3.2.2 Pricing
```

- **Personal:** FREE
- **Catalyst:** $25+ (one time payment)
- **Commercial:** $50/month per user
- **Add-ons:**
  - Sync $4
  - Publish $8

Obsidian's starting plan is free, which stands out compared to the lowest price tag of Roam Research ($15).

```
3.2.3 Security
```

For your safety, **Obsidian Sync** encrypts your **remote vault** and all communication with Obsidian's servers. Before anyone can access your remote vault, they first need to decrypt it with an *encryption password*.

When you create a new remote vault, you have two options:

- **End-to-end encryption:** Obsidian encrypts the vault on your device with a custom encryption key before it's sent to Obsidian's servers. This guarantees that no one—not even the Obsidian team—can access your notes.
- **Managed encryption:** If you don't want to have to remember another password, you can let Obsidian manage the encryption password for you. While we store your encryption password on our servers, we only use it to offer a more convenient way to manage your vaults. Obsidian will never access your vault without your explicit consent.

If you forget or lose your custom encryption password, your data remains encrypted and unusable forever. We're not able to recover your password, or any encrypted data for you.

Your choice only affects your remote vault. Obsidian does not encrypt your local vault.

4. Supporting Information

Obsidian is in public beta right now. We have **a roadmap** that you can check out.
Our Twitter handle is **@obsdmd**, feel free to follow. We mostly tweet about product updates.
We have active **Discord** and **Forums**, and the **community** is generally quite helpful.

Experiment 5

Introduction to UML and its Tools

Unified Modeling Language (UML)

- UML version 1.1 was adopted in November 1997 by the Object Management Group (OMG) as a standard language for object-oriented analysis and design. Initially based on a combination of the Booch, OMT (Object Modeling Technique) and OOSE (Object-Oriented Software Engineering) methods, UML was refined and extended by a consortium of several companies, and is undergoing minor revisions by the OMG Revision Task Force. It was initially started to capture the behavior of complex software and non-software system and now it has become an OMG standard.

- The UML is a standardized general-purpose visual modeling language for specifying, constructing, visualizing, and documenting the software system and its components. The UML is a graphical language with sets of rules and semantics. The rules and semantics - of a model are expressed in English in a form known as OCL (Object Constraint Language). OCL uses simple logic for specifying the properties of a system. The UML is not intended to be a visual programming language. However, it has a much closer mapping to object-oriented programming languages.

Conceptual Model Of UML

The conceptual model of UML can be mastered by learning the following three major elements−

- UML building blocks
- Rules to connect the building blocks
- Common mechanisms of UML

UML Building Blocks

The building blocks of UML can be defined as −

- Things
- Relationships
- Diagrams

Things:

Things are the most important building blocks of UML. Things can be −

- Structural
- Behavioral
- Grouping
- Annotational

Structural Things
Structural things define the static part of the model. They represent the physical and conceptual elements. Following are the brief descriptions of the structural things.
Class − Class represents a set of objects having similar responsibilities.
Interface − Interface defines a set of operations, which specify the responsibility of a class.
Use case −Use case represents a set of actions performed by a system for a specific goal.
Component −Component describes the physical part of a system.
Node − A node can be defined as a physical element that exists at run time.

Behavioral Things
A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things −
Interaction − Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.
State machine − State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change.

Grouping Things
Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available −
Package − Package is the only one grouping thing available for gathering structural and behavioral things.

Annotational Things
Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.
Relationship − Relationship is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.
There are four kinds of relationships available.
Dependency − Dependency is a relationship between two things in which change in one element also affects the other.
Association − Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.
Generalization − Generalization can be defined as a relationship which connects a specialized

element with a generalized element. It basically describes the inheritance relationship in the world of objects.
Realization − Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.

UML Diagrams
UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete.
UML includes the following nine diagrams.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Statechart diagram
- Deployment diagram
- Component diagram

Rules to connect the building blocks
The UML's building blocks can't simply be thrown together in a random fashion. Like any language, the UML has a number of rules that specify what a well-formed model should look like. A well-formed model is one that is semantically self-consistent and in harmony with all its related models.

The UML has syntactic and semantic rules for

- Names- What you can call things, relationships, and diagrams
- Scope -The context that gives specific meaning to a name
- Visibility - How those names can be seen and used by others
- Integrity -How things properly and consistently relate to one another
- Execution -What it means to run or simulate a dynamic model

Models built during the development of a software-intensive system tend to evolve and may be viewed by many stakeholders in different ways and different times. For this reason, it is common for the development team to not only build models that are well-formed, but also to build models that are

- Elided- Certain elements are hidden to simplify the view.
- Incomplete- Certain elements may be missing
- Inconsistent- The integrity of the model is not guaranteed
  The rules of the UML encourage you but do not force you to address the most important analysis, design, and implementation questions that push such models to become well-formed over time.

Common Mechanisms Of UML
UML is made simpler by the presence of four common mechanisms that apply consistently throughout the language.

- Specifications
- Adornments
- Common divisions
- Extensibility mechanisms

Specification
In UML, behind each graphical notation, there is a textual statement denoting the syntax and semantics. These are the specifications. The specifications provide a semantic backplane that contains all the parts of a system and the relationship among the different paths.

Adornments
Each element in UML has a unique graphical notation. Besides, there are notations to represent the important aspects of an element like name, scope, visibility, etc.

Common Divisions
Object-oriented systems can be divided in many ways. The two common ways of division are-

Division of classes and objects − A class is an abstraction of a group of similar objects. An object is the concrete instance that has actual existence in the system.

Division of Interface and Implementation − An interface defines the rules for interaction. Implementation is the concrete realization of the rules defined in the interface.

Extensibility Mechanisms

UML is an open-ended language. It is possible to extend the capabilities of UML in a controlled manner to suit the requirements of a system. The extensibility mechanisms are −

Stereotypes − It extends the vocabulary of the UML, through which new building blocks can be created out of existing ones.

Tagged Values − It extends the properties of UML building blocks.

Constraints − It extends the semantics of UML building blocks.

UML Architecture
UML plays an important role in defining different perspectives of a system. These perspectives are −

1. Design
2. Implementation
3. Process
4. Deployment

The center is the Use Case view which connects all these four. A Use Case represents the functionality of the system. Hence, other perspectives are connected with use case.

- **Design** of a system consists of classes, interfaces, and collaboration. UML provides class diagram, object diagram to support this.

- **Implementation** defines the components assembled together to make a complete physical system. UML component diagram is used to support the implementation perspective.

- **Process** defines the flow of the system. Hence, the same elements as used in Design are also used to support this perspective.

- **Deployment** represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

Fields applying UML
UML has been used in following areas. UML can also be used to model non-software systems, such as workflow in the legal systems, medical electronics and patient healthcare systems, and the design of hardware.

Advantages Of UML

- You can model just about any type of application.
- UML is effective for modeling large, complex software.
- It is simple to learn for most developers, but provides systems.
- Advanced features for expert analysts, designers and architects.
- It can specify systems in an implementation independent manner

Disadvantages Of UML

- It takes a lot of time to keep the diagram reasonable and synchronized with the actual code. UML diagrams don't run, but require a lot of time. So they are good only if your

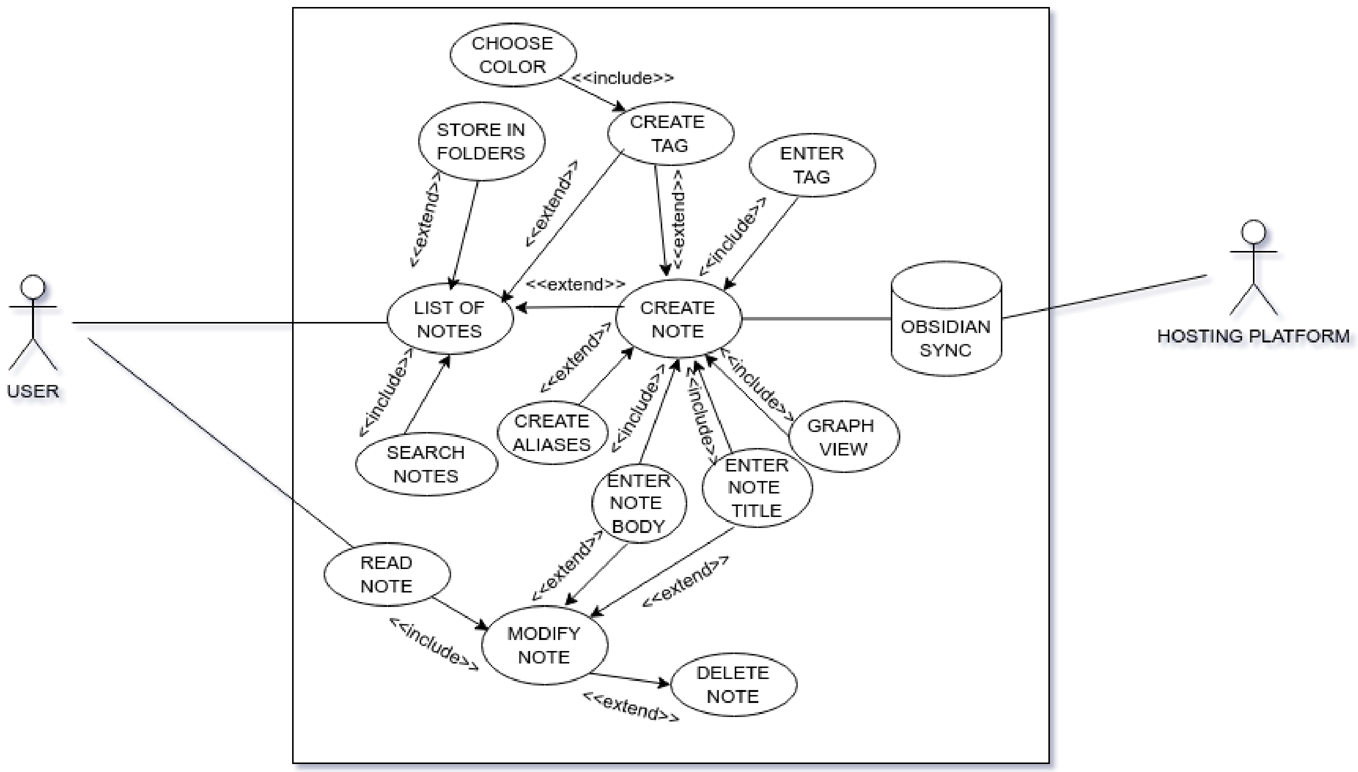
organization size can manage them.
- You cannot represent every condition in a sequence diagram. It's impossible if you want to deliver. So state diagrams should convey basic facts, not all the possible outcomes.
- Good UML software costs money and it takes some time to master properly.

Experiment 6

Use-Case Diagram

Use-Case Diagram: A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has, and will often be accompanied by other types of diagrams as well.

The greatest advantage of a use case diagram is that **it helps software developers and businesses design processes from a user's perspective**. As a result, the system functions more efficiently and serves the user's goals.



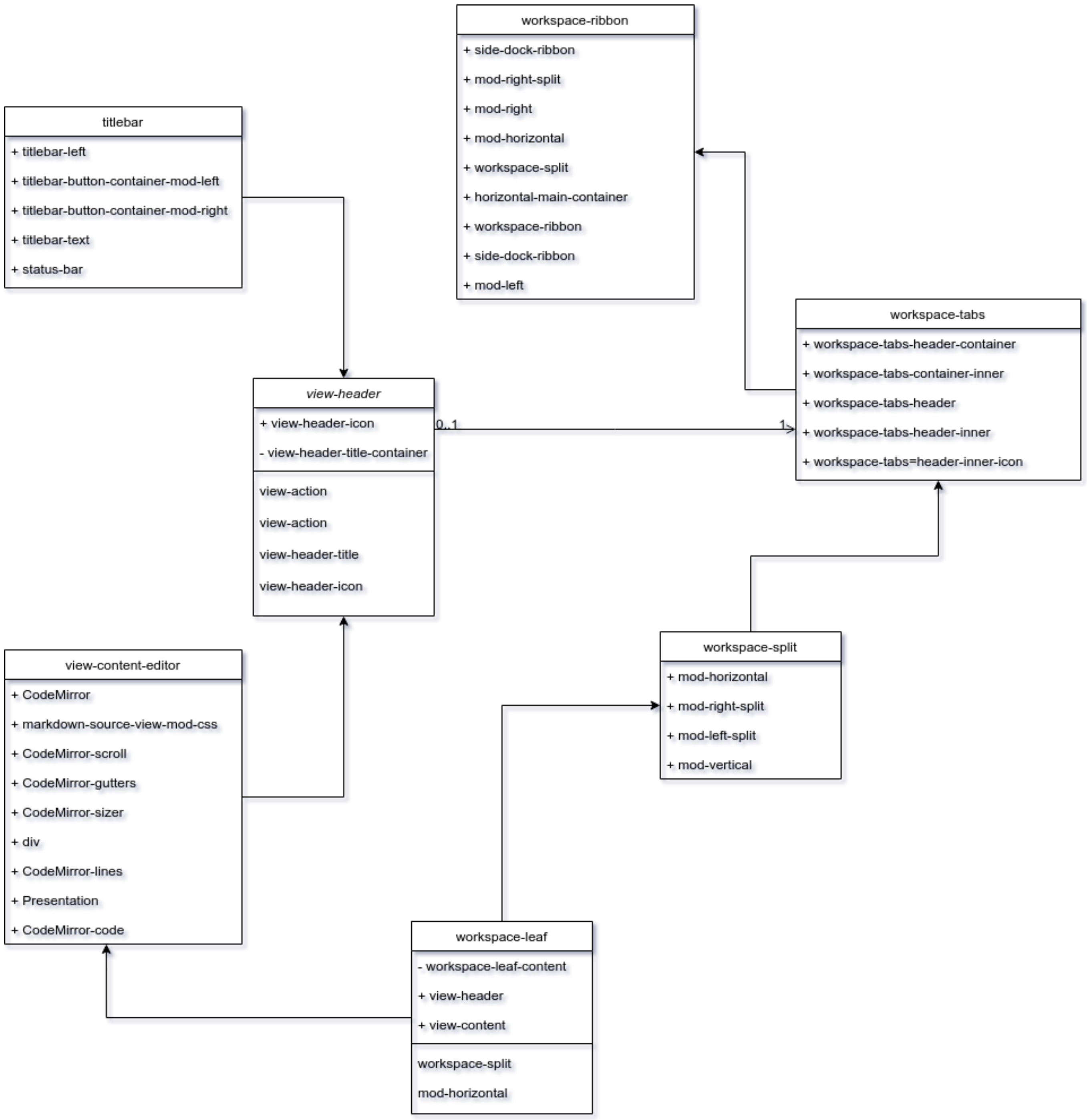
Experiment 7

Class Diagram

A class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects. In our diagram the class user and administrator is the combined login which leads us to the account. The account class can be used to view profile and jobs like sending friend request and reporting profile. Other than this, registrations are done on the account. Using profile class, we can send and delete messages and also requests the scheduler class to pass the message and other information including issues of the user. After the meeting being scheduled by the scheduler class, the therapist class has all the information related to the patient and they provide consultations, keep records of the patients and if the situation is out of hand, suicide prevention helpline is contacted at the earliest. Unlike the user whose identity is anonymous, the therapist is known and has a proper therapist id, name and email.

A class diagram could be implemented in different phases of a project and is the heart of the UML. A representation of reality is created by the class diagram by appearing on the domain model during analysis. The software modeling is done during the design phase, whereas the code is generated during the implementation phase. The foundation of software products is the class diagrams which are an essential part of any project.
A sense of orientation is given by the class diagrams. The structure of the system is analyzed in detail by the class diagram, and also the synergy among different elements is overviewed by them along with their properties. It is fast and easy to read and could be created easily if the right software is in place. Any system that needs to be created, the class diagrams form the foundation for that.

Class diagrams **give you a sense of orientation**. They provide detailed insight into the structure of your systems. At the same time they offer a quick overview of the synergy happening among the different system elements as well as their properties and relationships.


workspace-ribbon
+ side-dock-ribbon
+ mod-right-split
+ mod-right
+ mod-horizontal
+ workspace-split
+ horizontal-main-container
+ workspace-ribbon
+ side-dock-ribbon
+ mod-left
titlebar
+ titlebar-left
+ titlebar-button-container-mod-left
+ titlebar-button-container-mod-right
+ titlebar-text
+ status-bar
workspace-tabs
+ workspace-tabs-header-container
+ workspace-tabs-container-inner
+ workspace-tabs-header
+ workspace-tabs-header-inner
+ workspace-tabs=header-inner-icon
view-header
+ view-header-icon
- view-header-title-container
view-action
view-action
view-header-title
view-header-icon
0..1
1
view-content-editor
+ CodeMirror
+ markdown-source-view-mod-css
+ CodeMirror-scroll
+ CodeMirror-gutters
+ CodeMirror-sizer
+ div
+ CodeMirror-lines
+ Presentation
+ CodeMirror-code
workspace-split
+ mod-horizontal
+ mod-right-split
+ mod-left-split
+ mod-vertical
workspace-leaf
- workspace-leaf-content
+ view-header
+ view-content
workspace-split
mod-horizontal

Experiment 8

Activity Diagram

Activity Diagram: Activity diagrams are graphical representations of workflows of step-wise activities and actions with support for choice, iteration and concurrency. An activity diagram is essentially a flowchart that shows activities performed by a system.

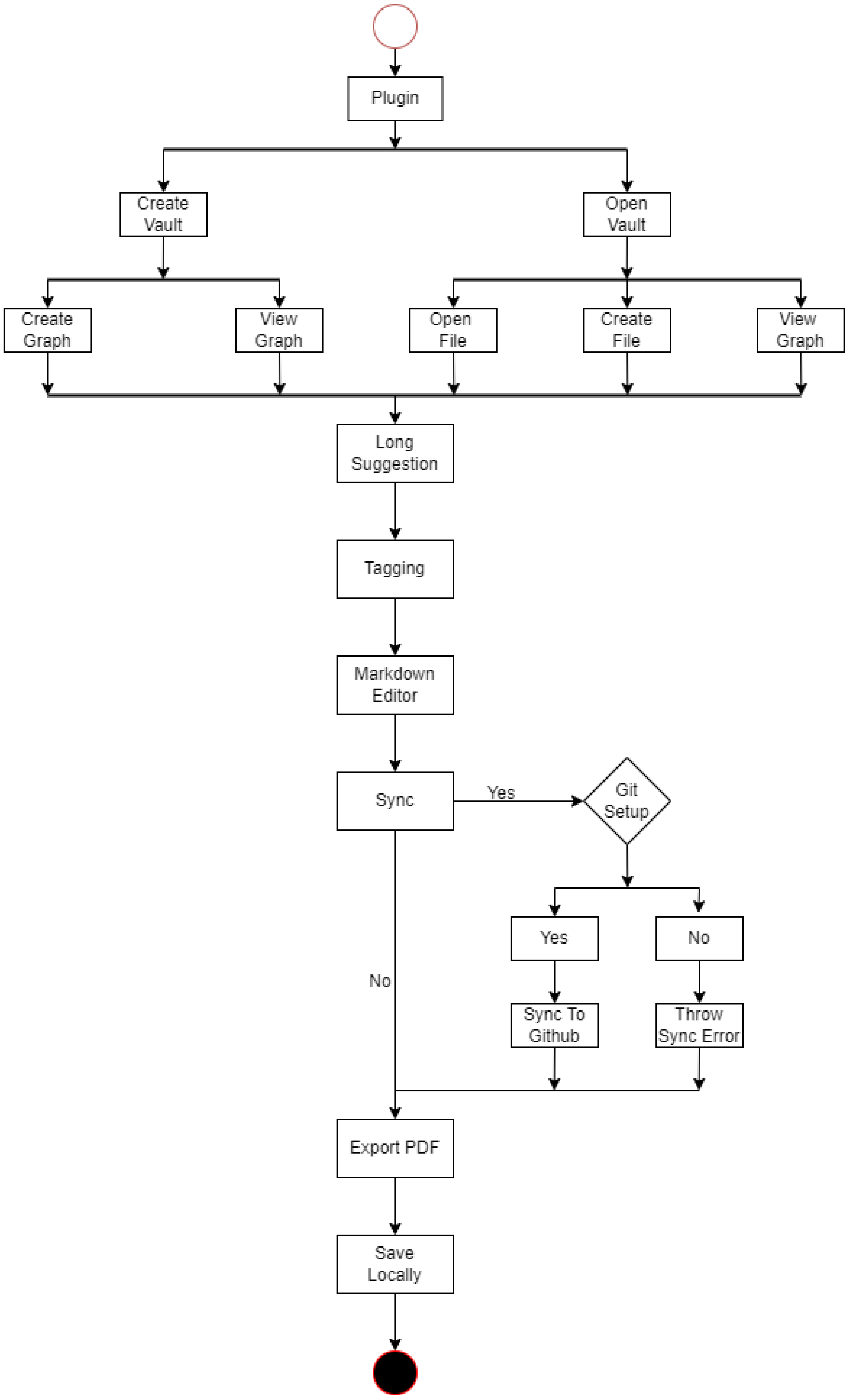
Activity diagrams can be used in many different situations. In addition, various relationships between activity diagrams and other UML diagrams can exist. Activity diagrams are well suited to visualizing models of procedures and their management. Activity diagrams illustrate the individual steps in activities as well as the order in which they are presented. They can be used for a range of functions: from the modelling of business processes all the way through to the depiction of control flows. Activity diagrams can be used anywhere where behavior needs to be described or where control flows need to be modelled.

An activity diagram shows business and software processes as a progression of actions. These actions can be carried out by people, software components or computers. Activity diagrams are used to describe business processes and use cases as well as to document the implementation of system processes.
Even the most complex progressions can be visualized by activity diagrams. Sequential and peripheral workflows are depicted by control and object flows. Activity diagrams represent activities that are made up by a flow of actions.

Activity diagrams area ideal for describing the following processes:

- Use cases and the steps described in them,
- Business processes or workflows among users and systems,
- Software protocol, i.e. the permissible sequence of interactions between components
- Software algorithms

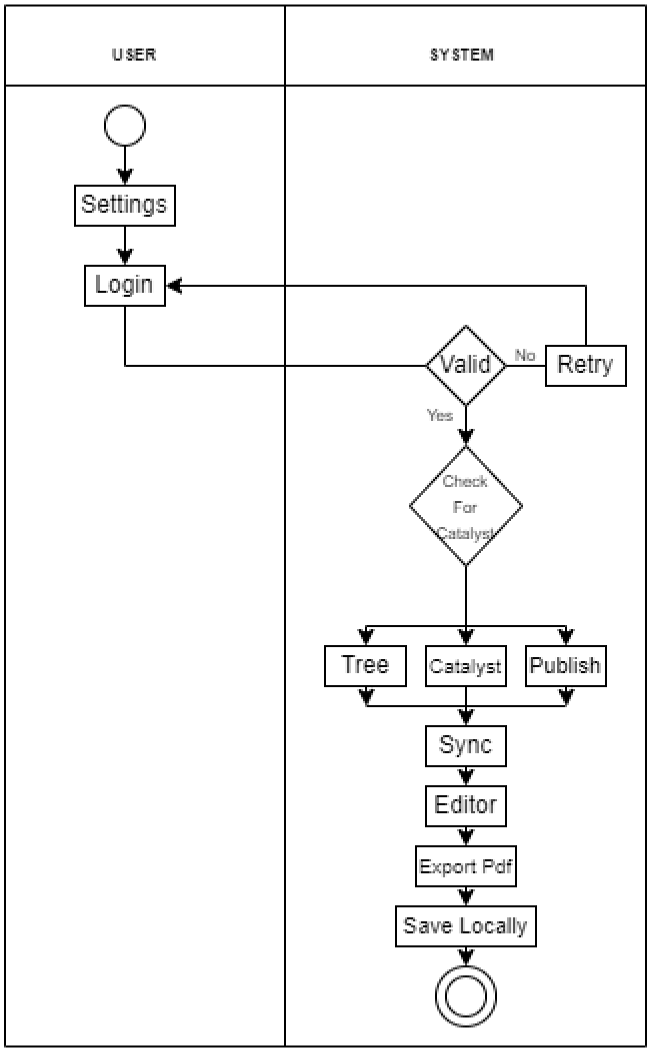
Activity diagrams can be divided into a few areas of responsibility such that these actions can be assigned to particular model elements like classes or components.


Plugin
Create Vault
Open Vault
Create Graph
View Graph
Open File
Create File
View Graph
Long Suggestion
Tagging
Markdown Editor
Sync
Yes
Git Setup
Yes
No
No
Sync To Github
Throw Sync Error
Export PDF
Save Locally

Experiment 9

Swim-lane Diagram

Swim Lane Diagram:- Swim lane diagrams are flowcharts that show both a process from start to finish and who is responsible for each step in the process.
Using the metaphor of lanes in a pool, a swimlane diagram **provides clarity and accountability** by placing process steps within the horizontal or vertical “swimlanes” of a particular employee, work group or department.
A swimlane (as in swimlane diagram) is used in process flow diagrams, or flowcharts, that **visually distinguishes job sharing and responsibilities for sub-processes of a business process**. Swimlanes may be arranged either horizontally or vertically.



Experiment 10

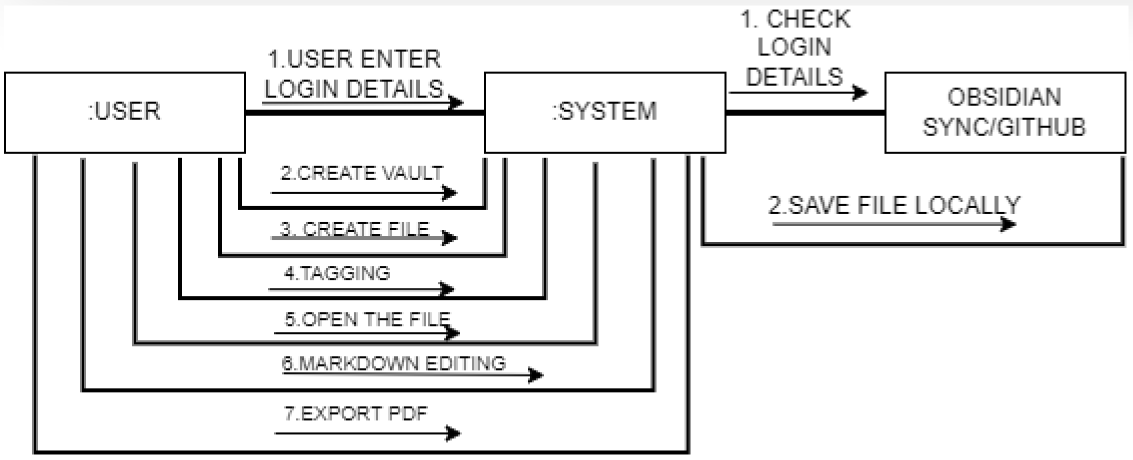
Collaboration Diagram

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). These diagrams can be used to portray the dynamic behavior of a particular use case and define the role of each object.

Our diagram represents sharing the health info and final records through communication controller to social network controller and eventually the external network. This is the communication medium starting from patient's report to final records to the respective network.

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML).

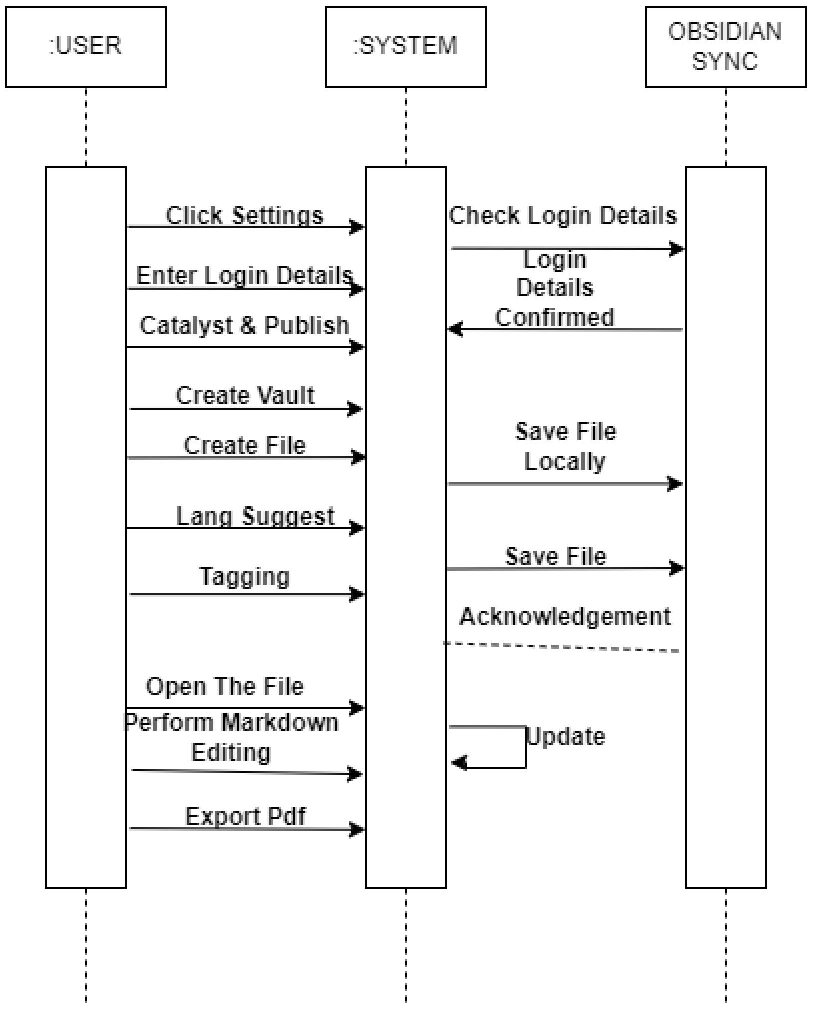
> *These diagrams can be used* ***to portray the dynamic behavior of a particular use case and define the role of each object****.*



Experiment 11

Sequence Diagram

A sequence diagram or system sequence diagram (SSD) shows object interactions arranged in time sequence in the field of software engineering. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. In our diagram, the user has to first go through the user interface for the status reporting. The user interface sends health information to the personal health information controller and requests data persistence to back-end, and the data is stored in the personal health information repository. Then the communication controller requests the repository for the health information and the information is sent to the controller. The health information is eventually sent to back end and the social network controller finally pass the information to the internal social network.

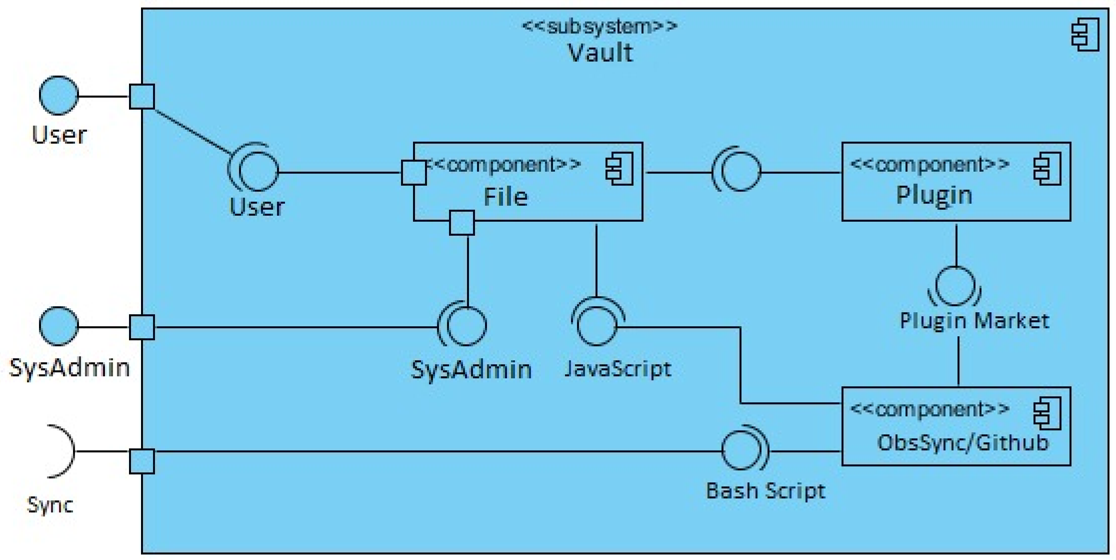


Experiment 12

Component Diagram

Component diagrams are used to visualize the organization of system components and the dependency relationships between them. They provide a high-level view of the components within a system. The components can be a software component such as a database or user interface; or a hardware component such as a circuit, microchip or device; or a business unit such as supplier, payroll or shipping.
Component diagrams:
• Are used in Component-Based-Development to describe systems with Service-Oriented-Architecture.
• Show the structure of the code itself.
• Can be used to focus on the relationship between components while hiding specification detail. • Help communicate and explain the functions of the system being built to stakeholders.

Component diagrams are **very simple, standardized, and very easy to understand**. It is also useful in representing implementation of system. These are very useful when you want to make a design of some device that contains an input-output socket.



Experiment 13

Deployment Diagram

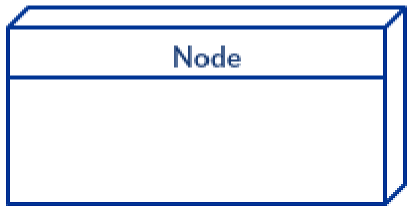
Introduction: A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them.

Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware.

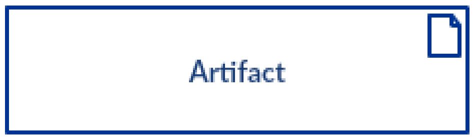
Deployment diagrams help model the hardware topology of a system compared to other UML diagram types which mostly outline the logical components of a system.

Deployment Diagram Notations: In order to draw a deployment diagram, you need to first become familiar with the following deployment diagram notations and deployment diagram elements.

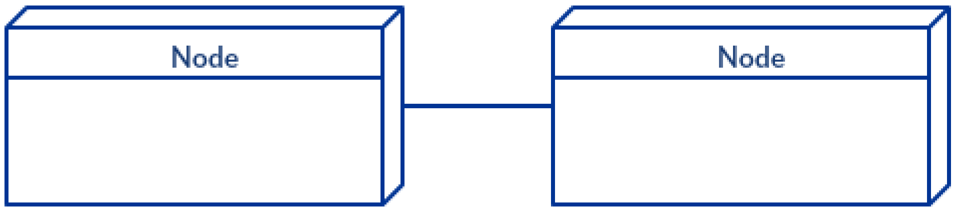
- Nodes: A node, represented as a cube, is a physical entity that executes one or more components, subsystems or executables. A node could be a hardware or software element.



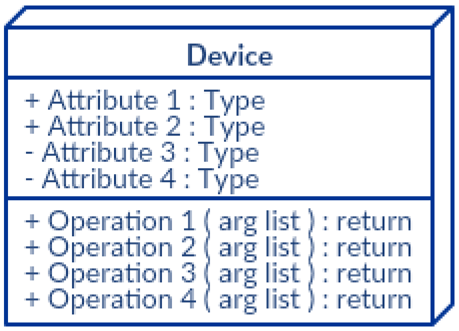
- Artifacts: Artifacts are concrete elements that are caused by a development process. Examples of artifacts are libraries, archives, configuration files, executable files etc.



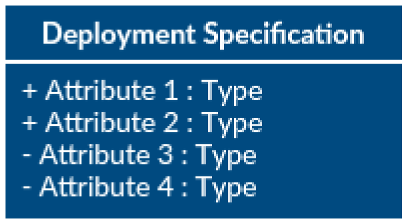
- Communication Association: This is represented by a solid line between two nodes. It shows the path of communication between nodes.



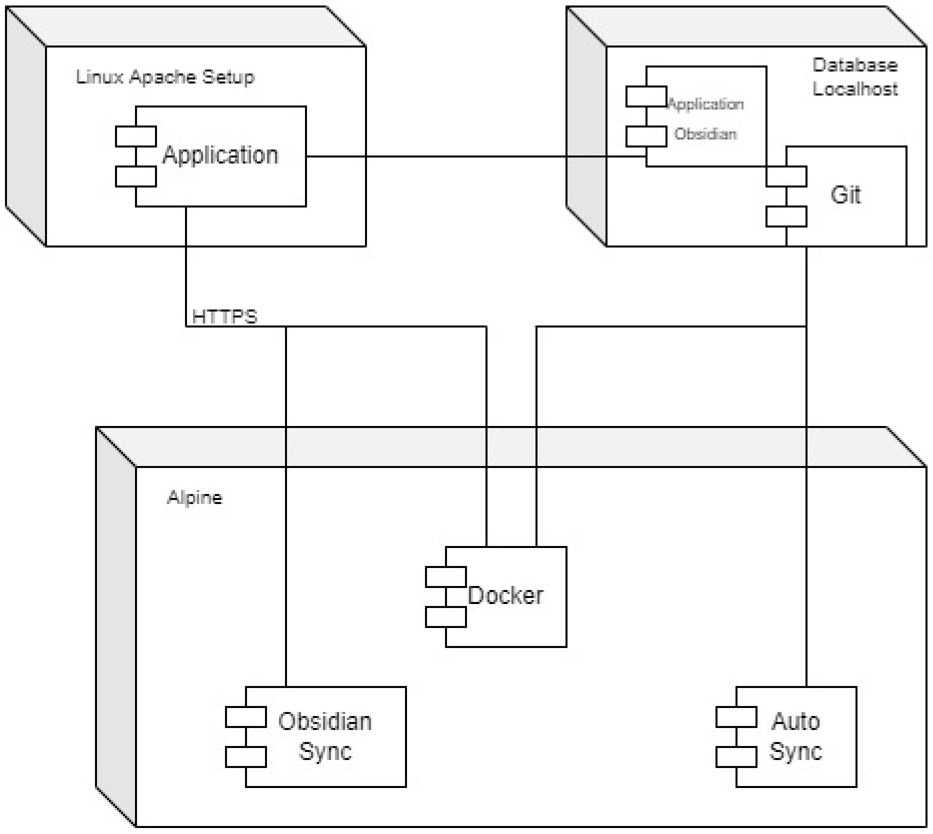
- Devices: A device is a node that is used to represent a physical computational resource in a system. An example of a device is an application server.


Device
+ Attribute 1 : Type
+ Attribute 2 : Type
- Attribute 3 : Type
- Attribute 4 : Type
+ Operation 1 (arg list) : return
+ Operation 2 (arg list) : return
+ Operation 3 (arg list) : return
+ Operation 4 (arg list) : return

- Deployment Specifications: Deployment specifications is a configuration file, such as a text file or an XML document. It describes how an artifact is deployed on a node.



Obsidian Deployment Diagram


Linux Apache Setup
Application
Database Localhost
Application
Obsidian
Git
HTTPS
Alpine
Docker
Obsidian Sync
Auto Sync

Experiment 14

Prepare Synopsis for suggested system

Abstract: "`OBSIDIAN- A second brain, for you, forever`". Obsidian is a powerful **knowledge base** on top of a **local folder** of plain text Markdown files, popular for **Back-Links**, **Graph-View**, **Extensively extensible**, and **Outgoing-Links** of documents.

There are various Markdown based Note-Taking applications are available in market. But none of them provide extended markdown feature with free-of-cost. Some are too heavy and have tons of features which we never gonna use. Having the huge variety of features consume high amount of memory which is not even required.

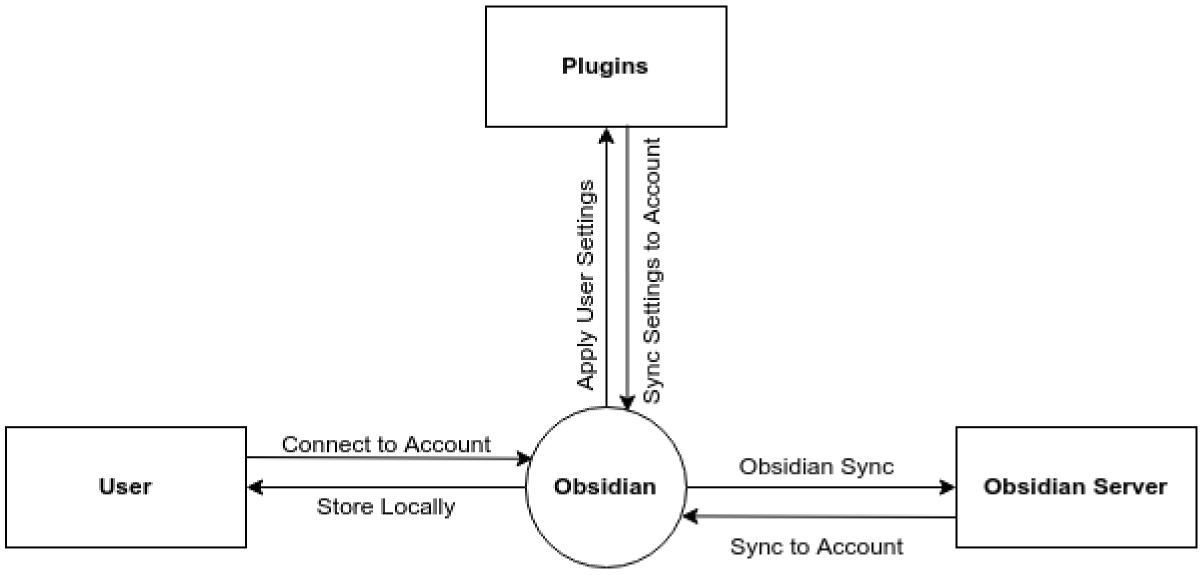
OBSIDIAN, provide complete markdown experience with built-in integration of mermaid along with back linking, graph view. Extreme lite application, which can even run smoothly on 15-year-old device. Minimalistic design of obsidian makes it more versatile and deduce scope bloat. A Hundred of handy features can add with the help of plugins.

Problem Statement: Notes are the important part of learning. Years ago, we make notes using COPY-PEN, but in that process we face many problems like bad-handwriting, damaged by external environment, can't access anywhere, connection between information is missing and many more... There are various NOTE-TAKING applications available in the market like - Microsoft 365, Gedit, Notion, EverNote, Google Keep, etc. Some applications are way-too-expensive, and not build for all the platforms. Some have funky animation which slows down the notes making process. Applications like Google Keep and Notion has some privacy breaches, that makes them unsafe in terms of privacy, and they are not super extensible. In these sense, Obsidian comes into the market.

Solution Statement: Obsidian is a powerful knowledge base on top of a local folder of plain text Markdown files. Markdown based multi-platform note-taking application free for all. It helps to connect information together, with the help of some tags. Suggest better words for your sentences and can predict next line with the help of plugins. All notes are in Markdown format, so you can easily export them in any format you want. Having a huge variety of useful plugin and themes. Live Preview feature for Markdown takes it to the next level. You can publish your notes/ documentation directly to the internet using this. Obsidian cares about your privacy. Account creation in not required until you purchase any plan. Version history is also supported using some extensions. Obsidian community has huge numbers of active helping members which are from different-different region and helps to resolve your problem

Introduction: With the help of this versatile application, we can easily choose the features we want with the help of extensions. Making notes using **OBSIDIAN** helps in connecting dots of information to make it knowledge. Memory consumption is low, hence it can be run smoothly on out-of-date devices. Storing data on a local device makes your notes more private and secure. You can purchase the subscription which cost less than a one time meal. With the help of subscription, you can sync notes on your all devices and your notes are encrypted in 48-bits `ENCRYPTION KEY` which can be only accessed by you.

Block Diagram:



Future Scope: In our age when cloud services can shutdown, get bought, or change privacy policy any day, the last thing you want is proprietary format and data lock-in.
With Obsidian, **your data sits in a local folder**. Never leave your life's work held hostage in the cloud again.

- **FUTURE-PROOF FORMAT**
  Obsidian uses Markdown. Markdown is widely used by sites like Reddit and GitHub and not going away anytime soon.
  Markdown is designed to be human-readable. It's not obfuscated by encoding; you can directly edit the source files with Markdown editors, or edit them automatically with scripts.

- **TOTAL CONTROL**
  Your notes live on your device, period. You can encrypt them or back them up however you want; it's your decision, not ours.
  Plain text files let you do various sync, encryption, and data processing on top of it.
  Obsidian plays nice with Dropbox, Cryptomator, and any software that works with plain text files.

+ **ALWAYS AVAILABLE**

No internet? No problem. Obsidian works completely offline, internet or service issues will never

be your problem.
Enjoy reading and working on your notes anytime, anywhere.

References:

Shida Li, Erica Xu, Sandy, Blaze (2020). "Obsidian Introduction". Retrieved from `https://obsidian.md/about`

Jeremy Valentine, **Phillip**, **Matthew Meyers**, **Ozan Tellioglu** (2020). "Obsidian Documentation". Retrieved from `https://github.com/obsidianmd/obsidian-releases`

Geeks For Geeks (2021). "Diagram References". Retrieved from
`https://www.geeksforgeeks.org/object-oriented-analysis-and-design/`,
`https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/`,
`https://www.geeksforgeeks.org/designing-use-cases-for-a-project/`,
`https://www.geeksforgeeks.org/class-diagram-for-library-management-system/`,
`https://www.geeksforgeeks.org/short-note-on-activity-and-swimlane-diagram/`,
`https://www.geeksforgeeks.org/component-based-diagram/`s