Licensing in Patient Monitoring System

A Thesis submitted in partial fulfilment of the requirement for the Award of the Degree of

MASTER OF TECHNOLOGY

in VLSI Design

Submitted By

Tanya Verma

602262024

Under Supervision of

Dr. Sanjay Kumar

Associate Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR INSTITIUTE OF ENGINEERING AND TECHNOLOGY

(A DEEMED TO BEUNIVERSITY), PATIALA, PUNJAB

JULY 2024

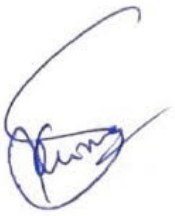
DECLARATION

I, **Tanya Verma** hereby declare that the work presented in this thesis entitled "**Licensing of Patient Monitoring System"** in partial fulfillment of the requirement for the award of degree of **Master of Technology (VLSI Design**) submitted at **Department of Electronics And Communication Engineering**, Thapar Institute of Engineering & Technology (Deemed to be University), Patiala is an authentic record of work carried out under supervision of **Dr. Sanjay Kumar** (**Associate Professor, Department of Electronics And Communication Engineering, Thapar University)** from **3 August 2023** to **30 June 2024**. The matter presented in this has not been submitted either in part or full to any other university or institute for the award of any other degree.

Date: 19/07/2024

Tanya Verma

602262024



(Mr. Manish Kumar Singh)

Manager

Date: 19/07/24

(Dr. Sanjay Kumar)
Associate Professor
Department of Electronics and Communication Engineering
Thapar Institute of Engineering & Technology, Patiala

Date: 19/07/24

PHILIPS INNOVATION CAMPUS

R T Nagar, Venkatala Village, Chowdeshwari Layout,

Venkatala, Bengaluru, Karnataka. 560064, India

CERTIFICATE

Date: 19th July 2024

This is to certify that Tanya Verma, a student of MTech (VLSI Design), Thapar Institute of Engineering & Technology, Patiala, is pursuing her internship program for 10 months (Aug 2023 to June 2024) of duration at **PHILIPS INNOVATION CAMPUS, BANGALORE**. Her title of dissertation is "**Licensing in Patient Monitoring System".**

Mr. Manish Kumar Singh

(Manager, HPM R&D Philips))

PHILIPS

Internship Certification

This is to confirm that

Ms. Tanya Verma

Has completed the project as an Intern at

Philips India Limited, Bangalore – Embassy Business Hub

From 03 August 2023 to 30 June 2024

Has worked on the project title

Synergy SoftOps

And completed the same satisfactorily

Reporting Manager: Manish Kumar Singh (70215915)

Place: Bangalore – Embassy Business Hub

Date: 28 May 2024

Siddhartha Choudhuri

Siddhartha Choudhuri

Philips People Services Manager India Hub

Philips India Limited

Philips Innovation Campus

Philips India Limited,
Embassy Business Hub, Venkatala Village,
Yelahanka, Bengaluru – 560064
www.philips.com, www.philips.co.in
Tel +91-80-4189 0000, CIN No: U31902WB1930PLC006663

Registered Office:
3rd floor, Tower A, DLF IT Park, 08
Major Arterial Road, New Town (Rajarhat),
Kolkata - 700156
www.philips.com
www.bangalore.philips.com

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people who made it possible whose proper guidance and encouragement has served as a beacon light and acted as a backbone for the completion of this project. I take an opportunity to thank all the distinguished personalities for their enormous and precious encouragement throughout this project.

I would also like to thank **Dr. Kulbir Singh**, Professor and Head of the Department of Electronics and Communication Engineering, TIET Patiala and for her continuous support and valuable suggestions during the project.

I want to profusely thank my guide **Dr. Sanjay Kumar**, Associate Professor, and all the other faculties from the Department of Electronics and Communication, TIET Patiala for their valuable guidance, regular source of encouragement and supervision throughout this project.

I owe sincere gratitude to Philips Innovation Campus India, my manager and mentor **Mr. Manish Kumar Singh**, for providing me with this wonderful opportunity to pursue my internship under his mentorship and my colleagues **Harsh Kumar Dedakiya, Abhay Mahajan,** and **Deewakar Shaw**, for their kind guidance and inspiration for learning and completing this project. I would also like to thank all my team members for their kind support throughout and for always helping me with all my doubts and queries.

I am indebted to all those who have directly or indirectly helped me in bringing out this project satisfactorily for the specified duration.

ABSTRACT

The implementation of the Patient Monitoring System (PMS) is very important for calculating the vitals in the Human body even in the dangerous situation so that PMS can measures the vitals properly and check the proper functioning of the Human body. Through the PMS many vitals can be measure such as blood pressure, body temperature, ECG, heart rate and SpO2.

Licensing plays an important role in the PMS for many reasons. Licensing maintains the standards while using inside the PMS and guaranteeing about the safety of a Patient body, Licensing not only crucial in the technology field but also equally important in the healthcare unit. Through Licensing developer's intellectual property rights can be protected.

In this Thesis the concern will be the product Licensing in the Patient monitoring Systems. Through Licensing there will be assurance in the standards, by proposing the feature-based Licensing model. After the product Licensing its testing is equally important, so functional testing comes into picture for the security and accuracy purpose. In this Thesis there are four functions which are important in the Licensing phase those functions are getfingerprint, installLicense, login and logout. Licensing gives the data privacy and system integrity.

By adopting the Licensing, we welcome the security and data privacy in the patient monitoring system by doing so trust among the patient, doctors, and stakeholders are maintained and by adopting Licensing patient care are improving.

TABLE OF CONTENTS

LIST OF FIGURES

Figures **Page No.**

LIST OF TABLES

Tables **Page No.**

Chapter–1

Introduction

The first and foremost aim of patient monitoring systems is to standardize the medical processes. By the help of product Licensing or more precisely feature-based Licensing developers' intellectual property rights will be safe and no other unauthorized technologies can use it by doing these new innovations and investments are not under some dangerous hands.

Securing product licensing for patient monitoring systems is crucial to guarantee compliance with regulations, accountability, and the safeguarding of intellectual property. This significantly enhances the overall reliability and safety of essential healthcare technologies.

Licensing involves one company granting another temporary access to its intellectual property, such as patents or trademarks, for a fee. The granting company is the licensor, and the recipient is the licensee. This arrangement facilitates business alliances and market entry by renting out product-based knowledge.

1.1 Overview of Licensing in Patient Monitoring

In this Thesis it will tell how Licensing helps the developers to be more trustworthy, in the healthcare industry everything should be transparent because it deals with the patient safety. The thesis will tell that Licensing protects the developers' intellectual property rights which promotes the new innovations and the investments in the monitoring industries. Thesis explains why Licensing is so important for the patient monitoring systems so that there should be no compromise for the patient safety devices.

For License a particular feature there are so many approaches. Health care market buy the Licenses for any feature and utilize it in the monitoring systems.

1.1.1 Flow of Licensing in Patent Monitoring system

In this project basically Licensing is done for feature present inside the Patient Monitor systems the steps for it are shown below:

1. Feature for a PMS.
2. Feature encrypted by License code.
3. Consumption of License Code in Patient Monitor System

1.1.2 Flow of Testing in Licensing

As we discussed above how Licensing happened in the patient Monitor systems, License Enforcement means testing of those functions which have been written for the feature present inside the Patient Monitor systems. The steps are shown below:

1. Unit Test cases with the help of GTest(GoogleTest)
2. Mocking of the Third-Party APIs using GMock(GoogleMock)
3. Check the coverage of the code.
4. Test the project on the hardware here in this project i.MX 8 Board has been used for testing.

1.2 Motivation

Philips which is the most renounced MNCs that works now under the health sector, so Philips goal is to prioritize the patient safety first by evolving the advanced technology keeping the quality in mind. Through so much research and development Philips ensures the patient's safety and quality even if it will take some time from medical imaging to monitoring devices even in the ultrasound sectors. Philips collaborates with the healthcare providers and agencies it advances its standards by winning the trust among them.

in this Thesis Product Licensing is very crucial in the software as well as in the technology side by safeguarding the intellectual property rights. Licensing establishes the legal frameworks to prevent unauthorized consumption especially in the data-sensitive sectors. There are so many models of Licensing such as one-time purchase or subscription which enhance the software quality and security. With the help of Licensing, it not only protects the data and security but also helps to increase the revenue of the companies.

1.3 Objective

Licensing requires adherence to certain quality standards and guidelines. This ensures that patient monitoring systems meet predefined benchmarks for accuracy, reliability, and safety. Quality control is critical in healthcare to prevent errors and ensure their well-being.

- The main objective of this project is to study the various strategies used in product licensing.
- Implement it in one of the features of a patient monitoring system used in healthcare industry.
- Various test cases are written to verify the functionality of the license.
- Verify the project on the hardware as well.
- Since licensing is a crucial part of a product, its functional validation becomes an important task

Chapter -2

Literature Survey

Software Licensing is very important for software development which gives several essential purposes. many numbers of research papers and conferences explore the important aspects of software licensing as it provides the tool for the software developers to increase the revenue of their development. [1]

In this research paper the study tells how Licensing is done in various technological fields including biotechnology, pharmaceuticals and even in semiconductors. It tells about the common Licensing technique like field-of-use restrictions, exclusivity clauses, and milestone payments used to control technology access. [2]

In this research paper it talks about the Licensing decisions from the firm's perspective by keeping the factors in mind such as market competition, technology maturity, and internal capabilities. In this paper it proposes the model for licensing the techniques like bargaining power analysis and joint ventures to navigate complex licensing scenarios and provides valuable insights into strategic licensing decisions.[3]

There is a comprehensive collection of patterns for designing and integrating appropriate license types into software products. The pattern language is accompanied by a detailed discussion of each pattern, including its context, problem, solution, and related patterns. This comprehensive approach provides software developers and vendors with a practical toolkit for making informed decisions about software licensing. [4]

Development of software is an intricate process that encompasses a wide range of activities beyond coding. Software engineering includes several tasks, including choosing and overseeing licensing for the specific project. The plugin searches the project for license information, as well as licenses for its libraries. If a license is already in place for the project, the plugin notifies the programmer of any potential license violations and offers recommendations to the developer for selecting the best open license. Utilizing Free Open-Source Software (FOSS) [5], [6], the issue of license infringement in these endeavors is growing in importance. Studies reveal that because open-source project developers frequently have limited time and resources and reuse a large amount of code, they may violate licensing agreements [7].

Recent years have seen several research about license violations in open-source projects [8, 9, 10]. This research showed that open-source projects have different degrees of potential license violations. This issue pertains to widely used licenses such as the restricted GPL-3.0 and the permissive Apache-2.0, as our earlier research [8] showed.

Software developers can benefit from using Free and Open-Source Software (FOSS), as they can reuse pre-existing FOSS to produce useful works [11]. Codes are made available and modified versions can be distributed under Open-Source Licenses (OSLs), which are used to award Open-Source Software (OSS). OSL offers software organizers intellectual property rights protection while luring contributors with similar open-source ideas. Software developers struggle to choose the right licenses for their open-source projects, even though open-source licenses (OSL) are crucial for project governance. Because it is difficult for them to recognize the differences in the legal phrasing of these license phrases and to comprehend the concept of licensing.

In this research paper Daniel etal. [12] conducted a check involving three popular open-source licenses, indicating legal pitfalls for lacking lores of OSL. Some experimenters tried to address this problem by developing license selection websites, similar as, TLDRLegal1, Choose license 2 and OSS- Watch3 etc. These tools do not directly give any guidance in suggesting licenses to inventors grounded on the features of stoner's systems.

In our research, software dependencies are considered when choosing a benefit license. Here, the dependency of the developer project (the third-party libraries) is input by the first process. By calculating the cosine similarity measure, this phase can locate related projects in the Maven repository. The second procedure determines whether the licensing of the third-party libraries the developers use in their projects are compatible with one another. This process is coupled with the algorithm that Kapisaki and Kramer et al. provided for the automatic determination of licensing compatibilities [13, 14]. Developers have access to a license list that complies with open-source component licenses. But we assert that the license selection model cannot replace legal counsel from attorneys or even come close to covering the intricacies of legal guidance.

Given the pervasiveness of digital technologies in contemporary culture, software plays an indispensable function. The terms of software use are outlined in software licenses, which are essential to the relationships between vendors and users. The Subscription model, which is based on cloud computing and SaaS principles, has been more popular recently among different licensing models. However, licensing itself is always changing, as are the validation processes that support it, like online activation and account control. The Subscription model is becoming more and more popular in modern software licensing, where customers renew their licenses on a regular basis to have access to the newest features. This strategy benefits merchants hoping for steady income and users looking for flexibility [15].

In contrast to software sales, licensing allows customers to purchase usage rights under predetermined guidelines. Oracle, a software seller, filed legal challenges against Used Soft, a German firm that invented the exchange of old software licenses [16]. The matter was brought before the Court of Justice of the European Union (CJEU), which ruled in Used Soft's favor in 2012. Based on the "exhaustion doctrine," the CJEU concluded that a copyright holder cannot object to the resale of a software license that has been sold, whether it be for digital or physical copies. Subdivision of volume licenses is still not allowed, albeit. [17] There are various licensing strategies such as product based, feature based etc. Software houses sell their products by transferring usage licenses of various software components to the customers. Depending on Software type, there are several different license types that allow controlled access of services. [18]

This article is devoted to the development of the key licensing system for applications. In the theoretical part of the article, there are certain cryptography methods, which can be used while implementing the licensing system. C++ can be applied for software licensing using various algorithms and you need to specify a folder location for the license file, your developer password string, the license key issued to the user and the name of the application. [19]

C++ is the programming language that will have the high understanding of the object-oriented programming, C++ is used when there are limited resources for the applications.[20]

Object-oriented programming gives the ideas of the classes and the four pillar of Oops such as inheritance, polymorphism, data abstraction, and encapsulation that will be helpful for the code reusability. [21]

C++ permits low-level data manipulation up to a point because it is tightly linked to C, a procedural language that is closely related to the machine language [22].

APIs, also known as Application Programming Interfaces, are among the advances in the software development field. An essential part of the software ecosystem are APIs [23]. Building complex software solutions on top of a shared technology platform has become incredibly easy thanks to these software ecosystems [24]. The main purpose of APIs was to facilitate communication between two or more programs (IBM, 2016). However, it was not until about the year 2000 that APIs on the web—more often known as web APIs—started to appear.

In this research paper Niu et al. [25], APIs helps in the software development and through APIs we can use the functionality of the different application can use inside our current applications. Similarly, Qiu et al. [25] says the same thing.

Software testing is a procedure that entails running a program or application and looking for any faults or bugs to produce software that is free of defects. Software testing is the only way to determine the quality of any software [26].

Iqbal Singh and Rasneet Kaur Chauhan talked about several software testing methods and instruments that are still in use today [27]. Software testing techniques like white box and black-box testing are explained, along with strategies like unit testing, which tests the program's fundamental unit, integration testing, which is carried out after the software has been integrated, and system testing, which tests the system [28].

Dr. N. Samba Siva Rao and P. Suresh Kumar demonstrated several software applications that have embraced agile development in their development. They also emphasized the significance of testing. According to them, the testing team's organization will be different in this instance than it would be in a regular setting, and all team members must support quality infusion. Additionally, they discussed how choosing an automated test tool and automating tests will enable the project's teams to do the work faster and more efficiently [29].

Vishawjyoti and Sachin Sharma talked about the value of testing in the process of developing software. The writers talked about the benefits and drawbacks of various testing methods. The web testing tool Selenium was their primary goal. How it operates and how the findings are stored in the background while the web application is being tested [30].

Using software testing techniques, Priyanka Rathi and Vipul Mehra presented their analysis of both automation and manual testing. They also discussed the value of testing in the IT industry and how it reveals flaws or errors in software development. They talked about the advantages of automation testing as well as the drawbacks of manual testing. They also covered automation testing and manual testing. Additionally, they discussed testing methodologies, automated software testing tools, developer-focused tools, functional testing tools, load testing tools, and the significance of metrics for assessing the product's quality and advancement [31].

Test cases helps to test the functionality of the functions is being tested and presented by Suresh Thummalapenta, Saurabh Sinha, Nimit Singhania, and Satish Chandra. They discussed how test cases that are written in natural language are frequently utilized as steps in industrial practice. They explained how the test cases are created and how they must be adhered to use their software application in any given situation [32]. Research by K. P. Jayant, Renu Garg, Vinod Kumar, and Prof. Ajaya Rana demonstrates that the software development life cycle (SDLC) phase of software testing is the most

K. Vijay and S. Baladwarakanath talked about the internal states of the software testing process, which contain several groups of function calls that are necessary for the testing process. To determine the

precise configuration, the internal state of the code is acquired. They employ both procedural and object-oriented programming structures in their internal state methodology [34].
Finding flaws in these atomic software components is the aim of unit testing. Test cases covering a certain procedure for a given set of input data are written for that reason. Following the execution of the code associated with a certain test case, the results received are compared to the expected values using various assert types [35].

Relying solely on real object implementations isn't always practical or convenient, especially in the early stages of testing and prototyping when it adds extra dependencies and needlessly complicates the program [36]. As a result, interface testing has existed alongside unit testing since its inception. This kind of testing is a variation on unit tests, with the primary objective being to ascertain whether the objects interact correctly with one another during method calls [37]. The so-called fake classes—whose objects mimic the genuine implementation but lack full functionality—are employed for this purpose.

Google's open-source Gtest framework was created with the goal of making the C++ programming language's unit test creation and execution more convenient [38].

GMock's purpose is to give test writers using gtest the tools to create mock classes that replicate specific aspects of the system being evaluated. Expectations in the context of interactions between the tested code and mock objects form the basis of its operation. The interface of these objects is identical to that of the system component they are meant to replace. The real component and the abstract class they implement are the same [39].

The relevant inputs obtained from Philips-specific document were also important for the smooth implementation of this project. [40]

Chapter 3
Infrastructure of Licensing in PMS

In the Monitoring systems, Licensing is the important intellectual property and maintained the standards. Through Licensing unauthorised access or usage can be prevented and ensure that integrity and security of the healthcare data will be maintained. Licensing also keeps the patient safety at first and promise the privacy in the medical environment. In this Thesis our focus will be on License Enforcement rather than the License Activation.

Licensing has two phases –

1. **License Activation**:

   License activation is the technique where enabling a software license for utilization, it involves the validation of the provided license key. After the activation the software becomes the operational means it can be used, by granting access to the features per the License agreement. This process ensures unauthorized access or the usage.

2. **License Enforcement:**

   License Enforcement involves the deploying of the License by looking into the License terms by monitoring and regulating the software consumption why regulating because it helps to check about the unauthorized access or the usage of that software that is exceeding the license terms, this will help to safeguard the developers' intellectual property rights.

3.1 License Models

In this Thesis, we talked about the Time-based and Perpetual Licensing both and how they deploy in the patient monitoring systems. Feature-based Licensing entitles a user to access an additional feature that isn't included with his or her user license, such as Marketing or WDC. Users will pay only for the utilize license that they will use not for the non-utilize license that they will not use this will helps in the cost of the software solutions whereas Perpetual Licensing means one-time payment for the software and provide the indefinite access to the purchased License version. This will help in the long-term cost predictability, avoiding recurring subscription fees, and empowers businesses to manage upgrades, offering a cost-effective option prioritizing stability and ownership of the software throughout its lifespan.

3.1.1 Perpetual Licensing

Perpetual Licensing is the simple and long-term cost-effective deployment model. Users or the consumers will make only one time payment and can the purchase version for a lifelong. Perpetual Licensing is different from the subscription model. Perpetual Licensing gives the ownership, stability and control on the upcoming upgrades or the updates. Perpetual Licensing offers Feature-based Licensing that provides flexibility and cost-effectiveness. Consumers that consume the License has to pay for those License that they want to access it will help to optimize the costs for the consumers and gives the particular resources what they want. With this Perpetual modelling business environments is drastically increasing and force to use the feature-based Licensing.

i. **Base Features** - It will be available or inbuilt inside the patient monitor without any Licensing and can be used by the Patient and the doctors without purchasing the License.
ii. **License Features** – These Features will be present in the Patient Monitors as per the doctor's requirements. If they need that feature, they will purchase the License and if they don't need it, they will not purchase it.

3.1.2 Time-based licensing

Time-based Licensing gives access to software for the specific time duration, unlike perpetual licenses, that allow indefinite access, time-based Licenses give temporary usage rights. consumers subscribe or rent software that is licensable for specific time duration only. Time-based Licensing is important for the short-term projects. In this Licensing long-term commitments are not there like in the perpetual Licensing this will ensure the efficient utilization of the software resources.

Benefits of using Time-based Licensing:

i. **Flexibility**: Provides short-terms access of the License, the consumers who need the temporary projects can use this.
ii. **Revenue Generation**: This will affect the business logic and overall revenue of the company because of the periodic renewal of the License.
iii. **Lower Entry Barrier**: It will make the products affordable which has this time-based License by segregating the cost for overall period.
iv. **Control & Versioning**: Through time-based Licensing helps to control the access of the specific versions of the software with its updates and upgrades.

3.2 License Type

There are so many types of License types like Node-Locked, Floating, Named User, Concurrent User, and Feature-Based licenses. Each one of these types has its own functionality. They give the flexibility in the business environment by looking into the consumption of the software Licenses in the

organizations by looking all the factors that how the License is consume by individual industry it will enhance the business of the organizations.

3.2.1 Concurrent License

In the concurrent Licensing multiple users can access the Licensed software at one time but the consumption of the License should be under the predefined limit means beyond that limit Industry or the organization cannot use that software. Concurrent Licenses are not like the user Licenses, that gives authority to the specific users only, concurrent Licenses gives the freedom that industry or organization can share the same software among the different users. We can take an example that if a company holds ten concurrent licenses, up to ten users can access the software simultaneously but if the eleventh user tries to access that license it will be denied for the eleventh user until the existing user logs out. This model optimizes software utilization and helps in increasing the cost or the revenue. This concurrent licensing is useful or beneficial when the organization has fluctuating users or teams.

3.2.2 Feature-Based License

Feature-based Licensing gives the window that will help to access a particular feature which should be licensable even without buying the License for whole software that's means pay only for the License of the features that we required rather than the whole software in which where there is no need of some of the features. This will help the consumers too, and that will optimize the costs of the purchasers. It will also help in the business and will help to increase the growth of the developers as well.

3.3 Design Pattern

Design patterns contribute to the field of software development and give opportunity to various challenges, by providing the blueprints or the basic structure that will enhance the scalability, flexibility, and maintainability in codebases. Design patterns give the idea of the framework for structure the code base components correctly and for the developers' design patterns are very crucial so create their own code base. There are so many types of design patterns such as creational, structural, and behavioral types. In the creational patterns focus will be on optimizing the object creation. In the structural patterns it focuses on the easy system architecture. Behavioral patterns help to learn about the interaction within the object if it is complex. Through the proper design patterns code quality will be maintained and more maintainable software solutions come into picture.

There are 2 main type of design pattern such as

3.3.1 Singleton Design Pattern

Singleton Design patterns help to create a single object for every class that is written in the code base. It also promises that there should not be more than one object. This single object which was created in the Singleton Design Pattern has global access all over the code base. This approach is beneficial during

memory management and other resources can also be utilized effectively. This one object interacts inside the code wherever we require it so this object can also be declared as a shared object or a shared resource. Through this single object code modularity will increase and help to simplify the maintenance of the code base.

3.3.2 Factory Design Pattern

The Factory design pattern, a creational pattern, furnishes an interface for object creation without specifying concrete classes. It encapsulates creation logic, enabling subclasses to determine the type of objects created. This promotes loose coupling between client code and objects, enhancing flexibility and maintainability. Factories are often employed to handle object creation complexities or implement dependency injection, fostering scalable and modular code architectures. By centralizing object creation, the Factory pattern encourages code reuse, simplifies testing, and upholds the open/closed principle, establishing it as a foundational tool in software design for object instance creation.

3.4 Wrapper

Wrapper class can be defined as the class that wraps the functionality of the APIs or can be wrap another object. Wrapper class hides the functionality of the functions and helps simplify the use of the functionality. In this Thesis Wrapper class is used for the License implementation. Wrapper classes are created and implemented on the Third-Party APIs.

Benefits of general wrapper class:

i. It helps to hide the complex implementation of the functions from the users.
ii. It provides the interface so that interaction can be made among different objects.
iii. If there are the requirements to add the additional functionality, that can also we add.
iv. It will help to increase or improve the code maintainability and modularity.

3.5 APIs

APIs stands for the Application Programming Interfaces are the sets of rules that should be use in the software applications so that proper communication could take place through these APIs exchange of the information can also be happen. APIs are used in various software applications, there are no applications where the APIs are not used. one of the basic examples of the APIs in the daily like is sign-in with the Google so Google is not providing the whole functionality but instead it is providing the interface to use it functionality.

3.6 Third-Party APIs

Third-party APIs consists of some rules and regulations when they interact with the applications with the functionality that is provided by the third-party. Through these third-parties APIs developers use the functionality of these APIs into their applications so that they could not write the whole functionality from the beginning. Developers require some sort of credentials for the usage of these APIs. This will help the third-party providers to keep track on the consumption of the functionality. GMock, which is one of the testing frameworks, is used to test these kinds of APIs.

3.7 Following things need to be taken care according to the Coding Guidelines

There are several coding guidelines that should keep in mind while working in the Synergy, Synergy means where multiple teams or individuals unite and come up with an outcome instead of looking into the individual efforts or contributions Synergy like to know about the collective efforts and how it will impact the whole team.

1. Naming conventions
2. Code organization
3. User Defined Datatypes
4. Conversions
5. security
6. Style
7. Comments
8. Static Objects - singleton Objects
9. Aliases
10. Static code analysis
11. Code performance and its optimization
12. Oops (Object oriented programming)
13. Object Allocation
14. Use of preprocessor
15. Cmake build environment

3.8 Aliases

Aliases in the C++ are the alternatives names that were assigned to the existing datatypes. Aliases can be used as with the typedef keyword or by using the using keyword (this using keyword was introduced in C++11) so that code readability, maintainability, and portability should be maintained. Aliases are very useful when the datatypes are very complex so that we create the alias for that particular datatypes this will help to look the code cleaner as compared to before. Aliases has the power to create the more meaningful names for the complex or custom data structures or we can also hide the implementation details of those complex data structures.

```cpp
#include <iostream>


void_t example()
std ::cout<< "I am an example"<<std :: endl;

int main()
{
    std::cout<<"Hello World";
    example();

    return 0;
}
```

Figure 3.1 – Code without the Alias

```
main.cpp:5:1: error: ‘void_t’ does not name a type; did you mean ‘void’?
    5 | void_t example()
      | ^~~~~~
      | void
main.cpp: In function ‘int main()’:
main.cpp:11:5: error: ‘example’ was not declared in this scope
   11 |     example();
      |     ^~~~~~~
```

Figure 3.2 - Output of Code without the Alias

In the above code, the datatype of function is void_t which is not any datatype so in the output it is showing an error that void_t does not name a type.

```cpp
#include <iostream>

using void_t = void;

void_t example(){
std::cout<<"I am an example";
}

int main()
{
    std::cout<<"Hello World"<<std::endl;
    example();

    return 0;
}
```

Figure 3.3 - Code with the Alias

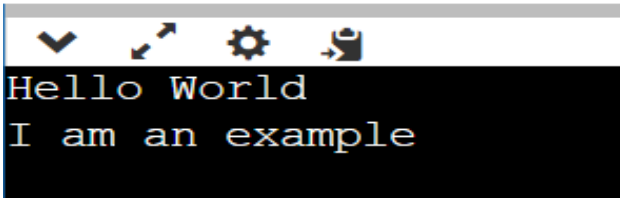


Figure 3.4 – Output of Code with the Alias

In this above code, the datatype of function is still the void_t but in the above line where the function is declared using keyword is used to specify the datatype, typedef can also be use in place of the using but according to the new C++ standard using is more appropriate as compared to the typedef.

3.9 Namespaces

In C++, namespaces are utilized to logically group code and alleviate naming conflicts throughout different sections of a program. They establish a designated scope, known as a namespace, where identifiers such as variables, functions, and classes can be defined. This structure assists in avoiding collisions between identifiers with identical names but originating from separate program segments. Employing namespaces aids in organizing code, enhancing code readability, and minimizing the potential for naming conflicts, particularly in extensive projects or when integrating multiple libraries. Namespaces in C++ enable the logical grouping of code elements to prevent naming conflicts and improve code organization.

```cpp
#include <iostream>

using std :: cout;
using std :: endl;

void example(){
    // using std :: cout;
    // using std :: endl;
cout<<"I am an example";
}

int main()
{
    // using std :: cout;
    // using std :: endl;
    cout<<"Hello World"<<endl;
    example();

    return 0;
}
```

Figure 3.5 – Code with using namespace (explicitly)

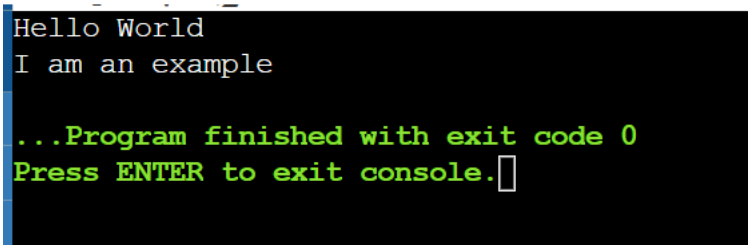


Figure 3.6 - Output with using namespace (explicitly)

In the above code**,** when we write using::std::cout, we are explicitly specifying that we want to use cout from the std namespace, but we are not bringing the entire std namespace into the current scope. Similarly, using:: std::endl; does the same for endl. This approach allows us to use specific names from a namespace without importing all its content.

```
1  #include <iostream>
2
3  // using std :: cout;
4  // using std :: endl;
5
6  using namespace std;
7
8  void example(){
9      // using std :: cout;
10     // using std :: endl;
11 cout<<"I am an example";
12 }
13
14 int main()
15 {
16     // using std :: cout;
17     // using std :: endl;
18     cout<<"Hello World"<<endl;
19     example();
20
21     return 0;
22 }
```

Figure 3.7 – Code with using namespace (implicitly)

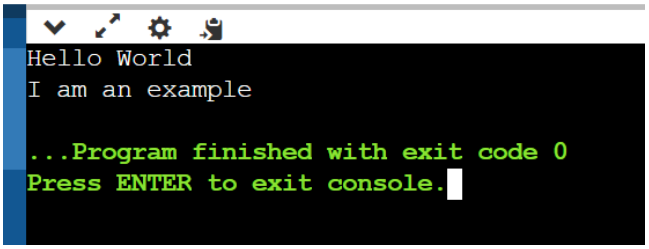


Figure 3.8 - Output using namespace (implicitly)

3.9.1 Create our own namespace

The process of creating a custom namespace in programming languages such as C++ or C# entails encapsulating code within a separate scope to avoid naming conflicts with other code. In C++, this is achieved by using the namespace keyword, followed by the desired namespace name, and enclosing the code to be encapsulated within curly braces.

```cpp
#include <iostream>

//creating the namespace
namespace MyNamespace {

    void myFunction() {
        std::cout << "This is a function inside MyNamespace" << std::endl;
    }
}


//using the namespace
int main() {
    // Using the function from MyNamespace
    MyNamespace::myFunction();

    // Alternatively, you can bring the entire namespace into scope
    using namespace MyNamespace;
    myFunction();

    return 0;
}

```

Figure 3.9 - Code for creating own namespace.

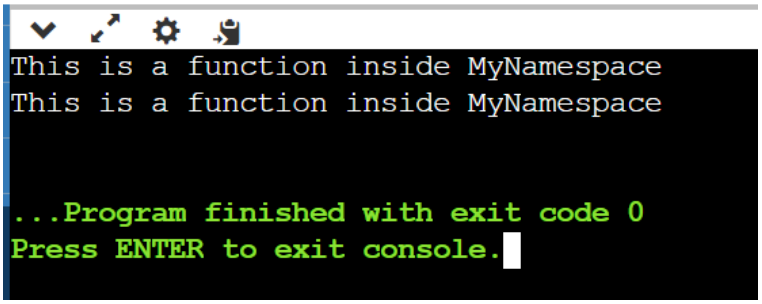


Figure 3.10 - Output of creating own namespace

This above code defines a custom namespace named "MyNamespace" containing a function called "myFunction ()". The function prints a message to the console. In the main () function, it calls myFunction () from the namespace using the namespace resolution operator. Additionally, it brings the entire namespace into scope using the "using" directive, allowing for direct access to the function without the namespace prefix. Finally, it calls myFunction () again outside of any function, demonstrating the usage of the function without the namespace prefix due to the earlier use of directive.

3.9.2 Error that will come if we use the using namespace.

Ambiguity error will arise.

```cpp
#include <iostream>

// Math library namespace
namespace MathLib {
    void calculate() {
        std::cout << "Calculating with MathLib..." << std::endl;
    }
}

// Physics library namespace
namespace PhysicsLib {
    void calculate() {
        std::cout << "Calculating with PhysicsLib..." << std::endl;
    }
}

int main() {
    // Bringing both MathLib and PhysicsLib namespaces into scope
    using namespace MathLib;
    using namespace PhysicsLib;

    // Attempting to call the calculate() function
    calculate(); // Which calculate() function will be called?

    return 0;
}

```

Figure 3.11 - Error coming while using namespace

```
input
Compilation failed due to following error(s).
main.cpp: In function 'int main()':
main.cpp:30:14: error: call of overloaded 'calculate()' is ambiguous
   30 |     calculate(); // Which calculate() function will be called?
      |     ~~~~~~~~~^~
main.cpp:19:10: note: candidate: 'void PhysicsLib::calculate()'
   19 |     void calculate() {
      |          ^~~~~~~~~
main.cpp:5:10: note: candidate: 'void MathLib::calculate()'
    5 |     void calculate() {
      |          ^~~~~~~~~
```

Figure 3.12 – Output of error coming using namespace

There are two different namespaces, MathLib and PhysicsLib, each defining a calculate () function. When there is use of using namespace MathLib and using namespace PhysicsLib, both namespaces are brought into scope. During calling of calculate () in the main () function, it's unclear which version of the calculate () function is referring to, as there are conflicting symbols in different namespaces. This ambiguity can lead to errors or unexpected behaviour. The compiler might not be able to resolve the call to calculate () correctly, leading to a compilation error or, worse, calling the wrong function unintentionally, which can result in runtime errors or produce incorrect results.

Note - To avoid such naming conflicts and potential ambiguity, it's best practice to use explicit namespace qualification or selective using directives to specify exactly which function you intend to use.

```cpp
#include <iostream>

// Math library namespace
namespace MathLib {
    void calculate() {
        std::cout << "Calculating with MathLib..." << std::endl;
    }
}

// Physics library namespace
namespace PhysicsLib {
    void calculate() {
        std::cout << "Calculating with PhysicsLib..." << std::endl;
    }
}

int main() {
    // Using the calculate() function from MathLib namespace explicitly
    MathLib::calculate();

    // Using the calculate() function from PhysicsLib namespace explicitly
    PhysicsLib::calculate();

    return 0;
}
```

Figure 3.13 – Code for resolving the issue.

```
Calculating with MathLib...
Calculating with PhysicsLib...


...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 3.14 - Output of resolving the issue

1. Explicitly qualify each call to the calculate () function with the namespace it belongs to (MathLib::calculate (), PhysicsLib::calculate ()). This approach ensures that there is no ambiguity about which version of the function is being called.
2. Avoid using using namespace directives to import entire namespaces into scope. Instead, we directly specify the namespace when calling the functions. This selective approach reduces the risk of naming conflicts and unintended behaviour.

By using explicit namespace qualification or selective using directives, we can clearly specify which function we intend to use, thereby avoiding naming conflicts and potential ambiguity, and ensuring more maintainable and predictable code.

3.10 CMake build Environment

CMake is the open-source and cross-platform building environment that helps to build the process or code for software projects. It creates the Make files automatically by looking into the specified compilers availability. Through the CMake build environment compilation and building done consistently in the various types of operating systems Developers create the language independent cmakeLists.txt file where all the details related to the project were kept and also it will tell how the project will build so by CMake build environment most of the work done automatically and it also help to optimize the development by ensuring the portability and scalability of the projects.

3.10.1 Creating the CMake build Environment

With the following steps, creation of a CMake build environment for the project, allowing for consistent and efficient cross-platform builds.

i. **Create a CMakeLists.txt file**: In the root directory of the project, create a CMakeLists.txt file. This file will serve as the main configuration script for CMake.
ii. **Define project details**: Inside the CMakeLists.txt file, use the **project ()** command to define basic project details such as its name and version.
iii. **Specify source files**: Use commands like **add_executable ()** or **add_library ()** to specify the source files that constitute project's targets (executable programs or libraries).
iv. **Declare dependencies**: If the project depends on external libraries, use commands like **find_package ()** or **target_link_libraries ()** to declare and link these dependencies.
v. **Set compiler options**: use commands like **set ()** or **target_compile_options ()** to specify compiler flags and options.
vi. **Generate build files**: Run CMake in the build directory of your choice, specifying the path to the directory containing the CMakeLists.txt file. This generates platform-specific build files (e.g., Makefiles, Visual Studio project files).
vii. **Build the project**: Use the generated build files to compile and build the project using the preferred build tool (e.g., make, Visual Studio).
viii. **Execute tests and install (optional)**: Optionally, set up tests using CTest and install targets using commands like **add_test ()** and **install ()** within your CMakeLists.txt file.

3.10.2 Advantages of CMake build Environment.

CMake offers a powerful and versatile build environment that simplifies the development process, enhances project scalability and maintainability, and promotes cross-platform compatibility, making it a popular choice for building software projects of all sizes and complexities.

The advantages of using a CMake build environment include:

i. **Cross-Platform Compatibility**: CMake generates platform-specific build files, allowing developers to build their projects consistently across different operating systems and development environments.
ii. **Simplified Build Process**: CMake abstracts away platform-specific build details, simplifying the build process and reducing the complexity of managing build configurations.
iii. **Scalability**: CMake supports large and complex projects with multiple source files, libraries, and dependencies, enabling developers to manage project growth effectively.
iv. **Dependency Management**: CMake facilitates dependency management by providing mechanisms to find, link, and include external libraries and dependencies seamlessly into the build process.
v. **Modularity**: CMake promotes modularity by allowing developers to define targets (e.g., executables, libraries) and organize source files and dependencies into logical units, enhancing code maintainability and reusability.
vi. **Flexibility**: CMake offers flexibility in build configurations, allowing developers to specify compiler options, linker settings, and other build parameters tailored to their project requirements.
vii. **Integration with IDEs**: CMake integrates well with various Integrated Development Environments (IDEs) and build tools, providing a cohesive development experience for developers using different tools and workflows.
viii. **Community Support**: CMake has a large and active user community, providing access to extensive documentation, tutorials, and community-contributed resources to assist developers in adopting and using CMake effectively.

This is how CmakeLists.txt file looks like, and that typically expand and customize it according to the specific requirements of the project, such as adding more targets, configuring compiler flags, or linking external libraries.

```
# Define project details
project(MyProject VERSION 1.0)

# Specify minimum required CMake version
cmake_minimum_required(VERSION 3.10)

# Add executable target
add_executable(my_executable main.cpp)

# Set C++ standard
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED True)

# Optional: Add include directories
#include_directories(include)

# Optional: Add library dependencies
#find_package(MyLibrary REQUIRED)
#target_link_libraries(my_executable MyLibrary::MyLibrary)

```

Figure 3.15 – Sample of CMakeLists.txt file

This example includes the following components:

i. **Project Definition**: The **project ()** command defines the project name and version.

ii. **CMake Minimum Required Version**: The **cmake_minimum_required ()** command specifies the minimum version of CMake required to build the project.

iii. **Executable Target**: The **add_executable ()** command adds an executable target named **my_executable**, which is compiled from the **main.cpp** source file. You can add additional source files as needed.

iv. **C++ Standard**: The **set ()** commands set the desired C++ standard for the project.

v. **Optional Components**: Additional components, such as include directories or library dependencies, can be added as needed. These are commented out in the example but can be uncommented and customized for your project.

3.11 UML:

Unified Modeling Language (UML) serves as a standardized visual modeling language employed in software engineering to design, document, and communicate the structure and behavior of intricate systems. It offers a collection of graphical notations for depicting different facets of a system, including its structure, behavior, interactions, and architecture.

Key components of UML include -

i. **Class Diagrams**: Represent the static structure of a system by illustrating classes, their attributes, methods, relationships, and constraints.

ii. **Use Case Diagrams**: Depict the interactions between actors (users or external systems) and the system to describe its functionality from a user's perspective.

iii. **Sequence Diagrams**: Show the sequence of interactions between objects or components in a system over time, detailing the flow of messages exchanged.

iv. **Activity Diagrams**: Illustrate the flow of control or behavior within a system, describing the sequence of actions or activities performed.

v. **State Machine Diagrams**: Model the behavior of individual objects or system components by defining their states, transitions between states, and associated actions.

vi. **Component Diagrams**: Display the physical components of a system and their dependencies to represent the system's architecture.

vii. **Deployment Diagrams**: Depict the physical deployment of software components across hardware nodes, illustrating the distribution and configuration of the system.

UML functions as a shared language among software developers, analysts, designers, and stakeholders, enabling them to communicate and comprehend the structure and behavior of a system across its lifecycle. It fosters clarity, consistency, and scalability in system design, promoting collaboration and assisting in the analysis, design, implementation, and maintenance of software systems.
For this project Sequence Diagram is one of the main aspects and there is an online tool through which Sequence Diagram can be created. Ex – Mermaid.

3.11.1 Mermaid

Mermaid create diagrams and visualizations using text and code. Mermaid allows even non-programmers to easily create detailed and diagrams through the Mermaid Live Editor.

1. Easy to use
2. Save development of time
3. Open Source
4. Simplify the documentation
5. Help code Logic via Charts

Mermaid is the flowchart and diagram visualization tool based on JavaScript; with the help of Mermaid, we can draw any graph but the keen interest here is the Sequence diagram.

3.11.1.1 Sequence diagram –

A Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. Mermaid can render sequence diagrams.

Sequence Diagram 1

The participants or actors are rendered in order of appearance in the diagram source text**.**

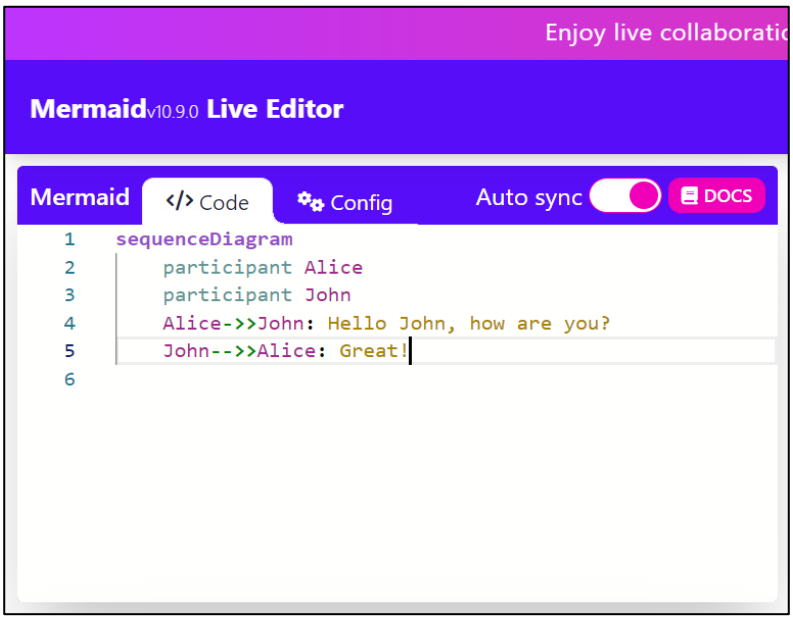


Figure 3.16 – Code of Sequence Diagram 1

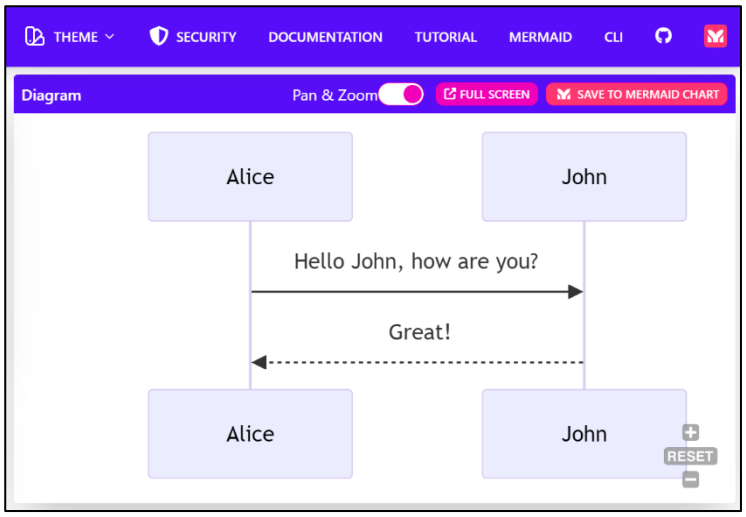


Figure 3.17 – output of Sequence Diagram 1

Note - The participants can be defined implicitly without specifying them with the participant keyword.

There are some messages that can be displayed. Messages can be of two displayed types either solid or with a dotted line.

Syntax - [Actor][Arrow][Actor]: Message text.

There are six types of arrows currently supported:

Arrow Type	Description
->	Solid line without arrow
-->	Dotted line without arrow
-»	Solid line with arrowhead
--»	Dotted line with arrowhead
-x	Solid line with a cross at the end (async)
–x	Dotted line with a cross at the end (async)

Table 3.1 Types of arrows use while creating Sequence diagram

Sequence Diagram 2

There is an option of activation and deactivation of an actor or the participants

Figure 3.18 – Code of Sequence Diagram 2

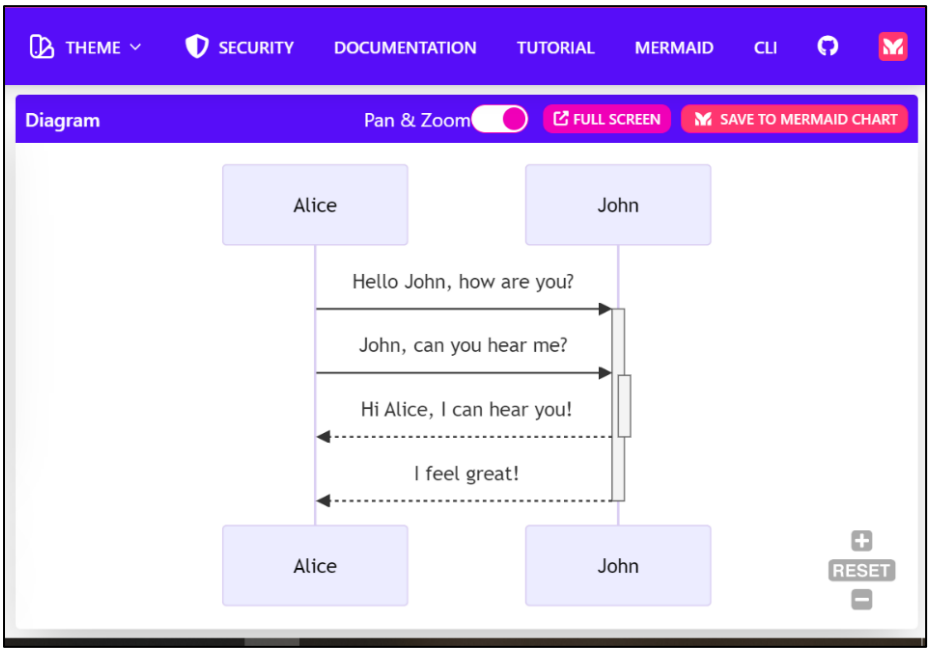


Figure 3.19 – Output of Sequence Diagram 2

i. Alice->>+John: Hello John, how are you?: This line represents a message being sent from Alice to John, indicated by an arrow pointing from Alice (Alice) to John (John). The ->>+ notation indicates that this is a synchronous message, meaning Alice waits for a response from John before proceeding. The message content is "Hello John, how are you?"

ii. Alice->>+John: John, can you hear me?: Like the first message, Alice sends another synchronous message to John, asking if he can hear her.

iii. John-->>-Alice: Hi Alice, I can hear you!: John responds to Alice's first message with a return message. The -->>- notation indicates that this is also a synchronous message. John acknowledges that he can hear Alice with the message "Hi Alice, I can hear you!"

iv. John-->>-Alice: I feel great!: John responds to Alice's second message, indicating that he feels great.

3.12 Unit Test Cases

Unit testing plays a crucial role in software development, focusing on testing individual units or components of a program to verify their correctness. For functions in C++ (or any other programming language), the creation of effective unit test cases is vital to validate their behavior and identify potential bugs early in the development cycle. Unit test cases can be developed using GTest, aiding in the creation of self-contained entities.

Unit Tests have some characteristics such as:

1. Run extremely fast (within the millisecond)
2. Run independently.
3. Doesn’t depend upon any external input.

There are various methods through which Test cases can be written from the various methods in this project focused will be on two testing framework such as Gtest(GoogleTest) and GMock(GoogleMock).

3.12.1 GTest (Google Test Framework)

Google test is a Xunit C++ testing framework. (Xunit means architecture of writing unit test code) Test framework means a software tool for writing and running unit tests. Portable and reusable i.e., same code can be used or works in Linux, windows, mac Libraries use in Gtest is- libgtest and libgtest_main. In windows if we are using VS, these libraries are self-integrated. In Visual Studio, precompiled header is usually named "pch.h". Apart from two libraries, there is a header file called #include<gtest/gtest.h> To run the GTest line should be present (testing::Initgoogletest (&argc, argv)).

When GTest is use for writing the unit test cases for the functions these things

i. To run the test case, test case is more like the function.
ii. To define the test case the syntax would be

TEST (test_name, subtest_1) these are the 2 arguments, there can be many sub test cases like subtest_2, subset_3.

3.12.2 GMock (Google Mock Test Framework)

GMock, also known as Google Mock, is a C++ framework developed by Google for writing and using C++ mock classes. It is part of the Google Testing Framework, which also includes Google Test for writing and running C++ tests. It is an extended version of the Google Test.

Google Mock is the extended version of the GTest in which we create the object of the mock class for testing. During creation of a mock object, define mock methods that allow you to specify expectations on how the methods will be called and what they should return. Creation of the Mock for the class is to provide the isolation from the external dependencies.

Mock method - A mock method is a key concept in the context of unit testing and the use of mocking frameworks like Google Mock in various programming languages. Mock methods are a way to simulate the behavior of a real function during testing, allowing to isolate the unit of the code that is under test by controlling the input and output of the mocked method.

Here are some important characteristics of mock methods:

i. **Simulate Real Methods**: Mock methods mimic the behavior of real functions without executing the actual code within them. This allows you to focus on testing the specific code you're interested in without worrying about the behavior of other components.

ii. **Behavior Control**: Expected behavior of a mock method, such as the return value or exceptions it should throw, in a controlled manner. This allows us to simulate various scenarios for testing.

iii. **Verification:** Mock methods also enable us to verify whether they were called with specific arguments and how many times they were called during a test. This is crucial for checking that code interacts correctly with its dependencies.

3.13 Code Performance

There are factors that should be kept in mind after the code implementation, which is the performance of the code. There are many performance parameters but here discussion will be on these two:

1. CPU Utilization
2. Memory

There are several commands for which Performance can be calculated such as Top and htop which are directly executed on the terminal with these commands performance of the binary file can be calculated.

3.13.1 Top -

The 'top' command functions as a system monitoring tool utilized in Unix-like operating systems to present real-time information about system resources, primarily focusing on CPU utilization. When invoked, 'top' dynamically displays active processes, offering insights into their CPU consumption. It arranges processes in descending order of CPU usage, enabling users to pinpoint those consuming the most CPU resources. Additionally, 'top' furnishes extensive data on system load averages, memory usage, and process statistics. Users can interact with 'top' in real-time, issuing commands to manipulate displayed information or adjust sorting criteria. This tool proves invaluable for system administrators and users troubleshooting performance issues, providing a swift and comprehensive overview of CPU utilization and process activity, thereby facilitating effective resource management and optimization endeavors.

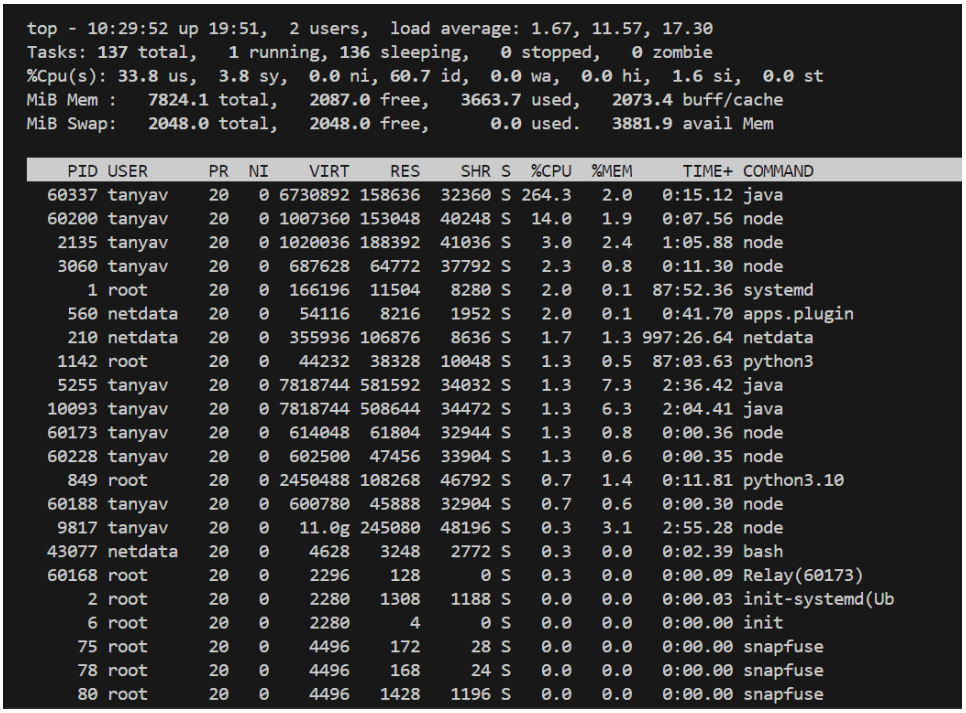


Figure 3.20 – Interface of the top command

In this Figure there are various parameters which tell about the CPU and Memory related to the project:

1. **Up -** It tell for how long the process is up.
2. **Users** – its value indicates the number of users currently logged into the system.
3. **load average** – It measures the system's overall workload and Average CPU load over the last 1, 5, and 15 minutes (reflects overall system demand).
4. **Tasks -** Total number of tasks among which how many are running, how many are sleeping, how many are stopped and how many are the zombie process which means processes that have finished execution but haven't been properly cleaned up yet and still occupying space and wasting resources.
5. **%CPU -** In the summary is the combination of all the cores CPUs. Top command does not talk about the how many cores are present. CPU has the 8 fields which tell more information related to the %CPU.
    i. **Us(user) -** This is the percentage of CPU, time spent running user processes, like the programs you're actively using.
    ii. **Sy(system) -** This is the percentage of CPU time spent running system processes, like background tasks.
    iii. **Ni(nice) -** This is the percentage of CPU time spent running processes with a lower priority (higher "nice" value). These processes typically have less impact on system performance and are given less CPU time when more important tasks are running.
    iv. **Id(idle) -** This is the percentage of CPU time that is idle, meaning it's not being used by any processes.

v. **Wa(iowait)** - This is the percentage of CPU time spent waiting for I/O operations to complete, such as reading or writing data to disk.

vi. **hi (hardware interrupts)** - This is the percentage of CPU time spent handling hardware interrupts, which are signals sent to the CPU by hardware devices.

vii. **si (software interrupts)** - This is the percentage of CPU time spent handling software interrupts, which are signals sent to the CPU by software within the operating system.

viii. **St(steal)** - It's a measure of time that a virtual machine (VM) is unable to use the CPU even though it's scheduled to do so.

6. **Memory usage** - measures in mebibytes (MiB), MiB Mem: "Mem" refers to physical memory (RAM).
    i. **total**: This shows the total amount of physical RAM available on our system.
    ii. **free**: This indicates how much physical RAM is currently unused and available for allocating to programs.
    iii. **used**: This shows the amount of physical RAM currently being actively used by running programs.
    iv. **buff/cache**: This refers to the amount of physical RAM used by buffers and caches. Buffers hold recently accessed data for faster retrieval, while caches store frequently used data to reduce disk access.
7. **Memory swap usage** - Reserved space on a hard disk or SSD acts as an extension of physical RAM. When RAM reaches its capacity, the operating system (OS) can temporarily transfer inactive memory pages to swap space to free up RAM for active processes. Although swap space increases memory capacity, accessing data from disk is much slower than from RAM, impacting performance.
    i. **total**: This indicates the total size of our swap space in MiB.
    ii. **free**: This shows how much of our swap space is currently unused and available for storing inactive memory pages.
    iii. **used**: This confirms that no part of our swap space is currently being used to store inactive memory pages.
    iv. **avail Mem**: This refers to the amount of available memory in our system, measured in MiB. This includes both free physical RAM and unused swap space.

Going on further, the detailed view of running process. There are so many columns which tell about the process information:

1. **PID (Process ID):** Unique identifier for each process.
2. **USER:** User who owns the process.
3. **PR:** Priority of the process (higher numbers mean lower priority).
4. **NI:** Nice value of the process (lower numbers mean higher priority.

5. **VIRT:** Virtual memory used by the process.
6. **RES:** Resident memory used by the process (actively in RAM).
7. **SHR:** Shared memory used by the process.
8. **S:** State of the process (R = running, S = sleeping, T = stopped, etc.).
9. **%CPU**: Percentage of CPU time used by the process.
10. **%MEM:** Percentage of memory used by the process.
11. **TIME+:** Total CPU time used by the process.
12. **Command:** Name or command line of the process.

3.13.2 Htop –

The htop command functions as an advanced and user-friendly substitute for the traditional top command, revered for its enhanced visualization of system resource utilization. Designed primarily for Unix-like operating systems, notably Linux, htop provides real-time insights into CPU, memory, and disk usage, featuring an intuitive color-coded interface that distinguishes between different processes and levels of resource consumption. Upon execution, htop displays an interactive and continually updating interface, enabling easy navigation through active processes, filtering based on criteria such as CPU usage or process name, and straightforward process management.

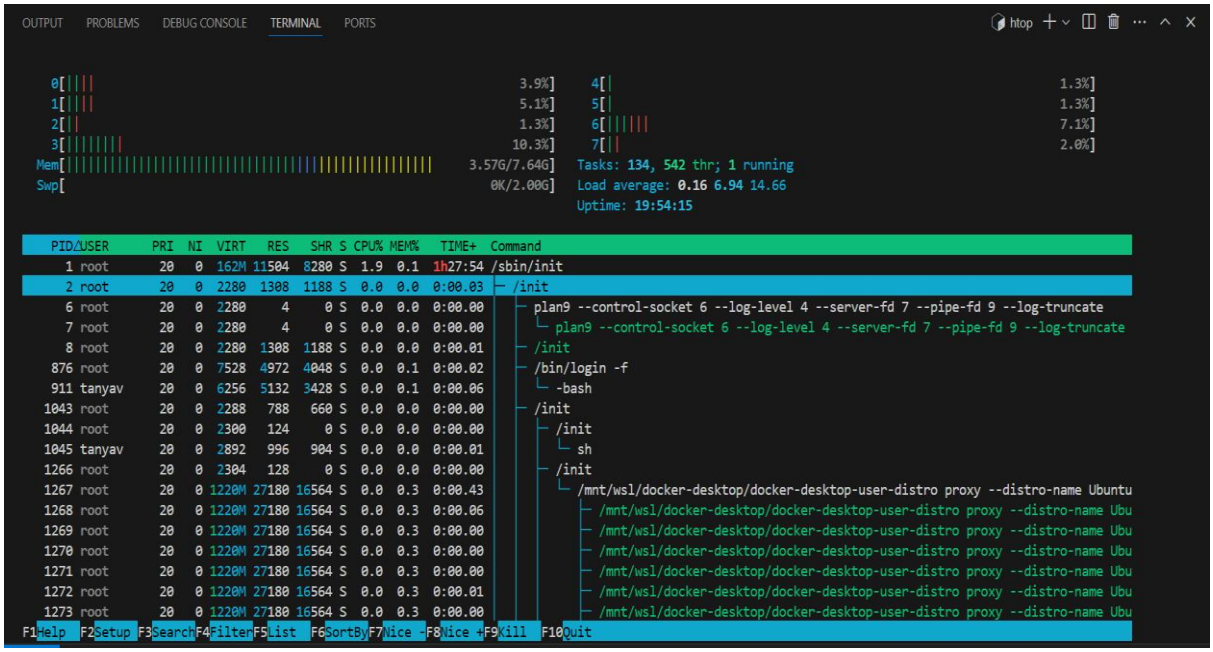


Figure 3.21 – Interface of htop command

In this Figure there are various parameters which tell about the CPU and Memory related to the project:

1. On the top window it talks about the summary in that the core usage separately present here, there are 12 cores showing the individual core usage in the visual representation.
2. In this core utilization the bars of the different colours which tells the core usage of the any process.

3. These coloured bars talk about the how much % of CPU core Utilization take place.
    i. Red bar shows it means very high utilization.
    ii. Yellow means the high utilization.
    iii. Blue means the moderate utilization.
    iv. Green means the very low utilization.
4. These values change after every 3 sec by default, but delay can be customized.
5. **Tasks** – It indicates the total number of processes currently running on our system.
6. **thr** - It refers to the total number of threads across all processes. Threads are smaller units of execution within a process, allowing for parallel or concurrent execution of tasks.
7. **Load average** – It will measure the system's workload over time. It shows the average number of processes that were either actively running on the CPU or waiting to run during specific time intervals.
8. **Uptime** – It talk about for how long the machine, or the system is active.
9. In the lower side of the htop command we have different functions keys like:
    i. **F1** - Access the built-in help documentation for understanding various features and commands within htop.
    ii. **F2** - Customize htop's appearance, behaviour, and display options:
    iii. **F3** - command helps to search the feature which we want to observe this feature is not observe on the top command.
    iv. **F4** - Filter the process list based on various conditions.
    v. **F5** - Toggle a tree-like view that visually groups processes based on their parent-child relationships, making it easier to understand process hierarchies.
    vi. **F6** - Change the sorting order of the process list based on different criteria.
    vii. **F7** - Lower the priority of selected processes, making them less likely to consume CPU resources when other processes need them, potentially improving system responsiveness.
    viii. **F8** - Raise the priority of selected processes, giving them more access to CPU resources, potentially improving their performance.
    ix. **F9** - kill the any process.
    x. **F10** - Exit the htop command and return to the terminal prompt.

3.14 i.MX 8 Board

The i.MX 8 series comprises a family of high-performance, energy-efficient system-on-chip (SoC) processors developed by NXP Semiconductors. These processors cater to a diverse array of applications.

Boards based on the i.MX 8 processors, often termed i.MX 8 boards, find widespread adoption among developers and manufacturers for prototyping, developing, and deploying embedded systems and IoT devices across various industries. These boards typically incorporate a variety of peripherals,

interfaces, and development tools to expedite prototyping and software development. The purpose of utilizing i.MX boards is to facilitate software development testing on the hardware.

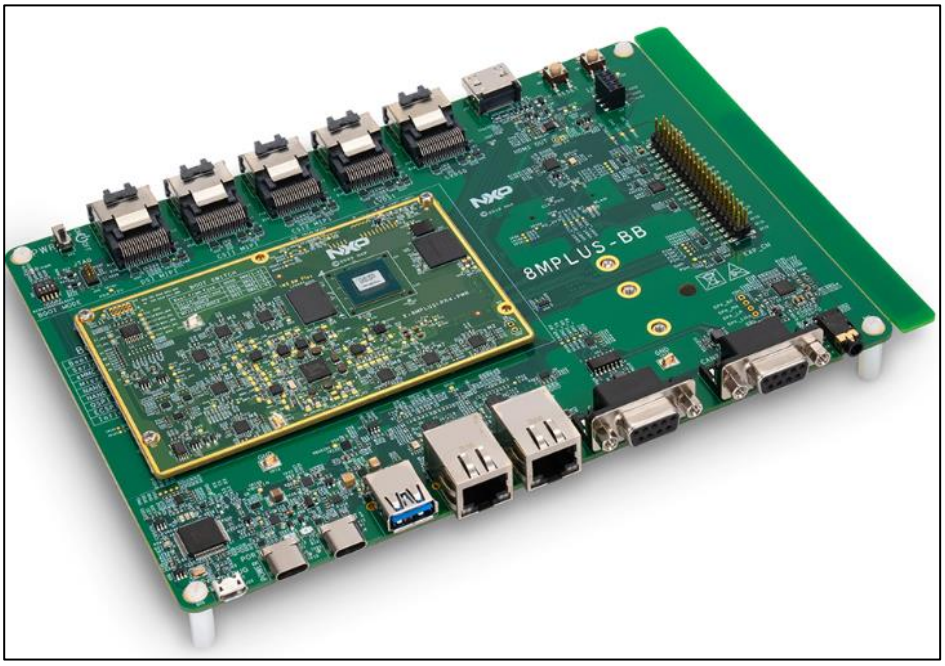


Figure 3.22 – i.MX 8 Board

3.15 SonarQube

SonarQube functions as an open-source platform for the continuous inspection of code quality. It employs static code analysis to automatically detect code smells, bugs, vulnerabilities, and security issues across various programming languages. SonarQube scrutinizes source code, pinpointing issues according to predefined rules, and furnishes detailed reports and metrics to aid developers in enhancing code quality and maintainability. Seamlessly integrating into the continuous integration and continuous deployment (CI/CD) pipeline, it enables teams to detect and rectify code quality issues early in the development cycle. Supporting an extensive array of programming languages, including Java, C/C++, C#, JavaScript, Python, and more, SonarQube emerges as a versatile tool for teams operating with diverse technology stacks.

3.15.1 Static code Analysis

Static code analysis, a method utilized to analyze source code without executing the program, involves examining the code structure, syntax, and semantics to identify potential issues, vulnerabilities, and code quality problems. Tools for static code analysis scan the source code, searching for patterns that may indicate errors, inefficiencies, or violations of coding standards. A wide range of issues, including coding errors, security vulnerabilities, performance bottlenecks, code smells, and adherence to coding standards and best practices, can be detected by these tools. Static code analysis assists developers in identifying and rectifying issues early in the development process before the code is executed or deployed. By providing feedback on code quality, security, and maintainability, static code analysis

tools enable developers to write cleaner, more robust, and secure code, ultimately reducing the risk of bugs and vulnerabilities, improving software quality, and enhancing the overall development workflow.

3.15.2 CI/CD Pipeline

The CI/CD pipeline, short for Continuous Integration/Continuous Deployment, is a set of automated processes that facilitate the development, testing, and deployment of software applications.

1. **Continuous Integration (CI) -** Continuous Integration (CI) entails the automated building and testing of code changes upon their commitment to the version control system. Developers frequently integrate their code changes (often multiple times a day), which triggers automated build and test processes. This guarantees smooth integration of changes, early detection of conflicts, and prompt identification and resolution of any issues.

2. **Continuous Deployment (CD)** - Continuous Deployment (CD) expands on CI by automatically deploying code changes to production or staging environments once they have successfully passed the automated tests. This facilitates swift and frequent software releases, diminishing the time and effort needed for deploying new features or bug fixes. Continuous Deployment pipelines typically involve supplementary procedures such as integration testing, user acceptance testing (UAT), and automated deployment to production environments.

Through the implementation of a CI/CD pipeline, development teams streamline the software delivery process, enhance collaboration between developers and operations teams, decrease manual intervention, and expedite the time-to-market for software updates. Furthermore, CI/CD pipelines contribute to the stability, reliability, and quality of software releases by automating testing and deployment processes.

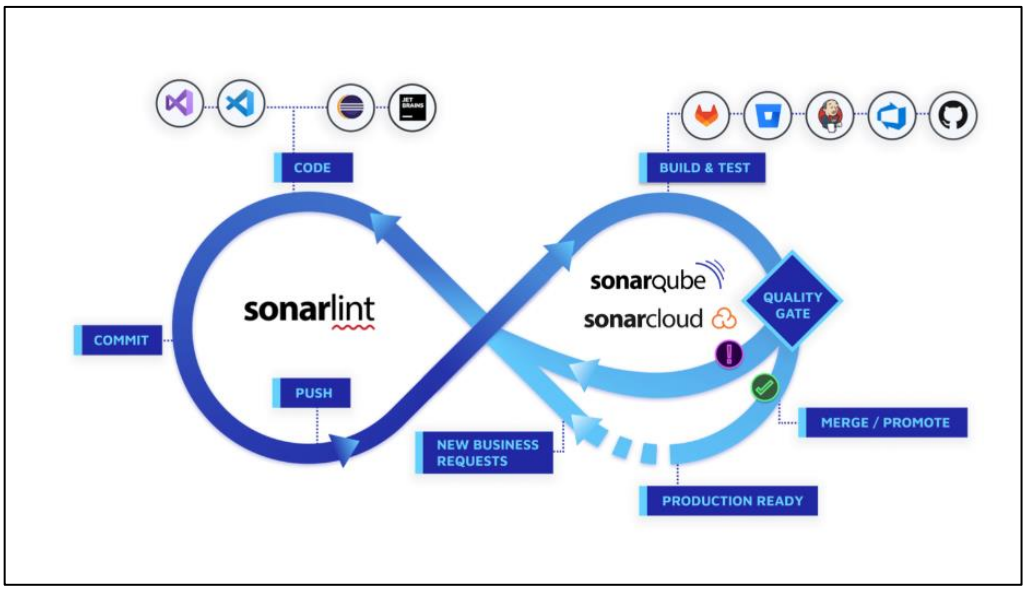


Figure 3.23 - Developing with Sonar

3.15.3 Parameters in SonarQube –

SonarQube analyses code and provides various parameters and metrics to assess the quality and health of the codebase. Some of the key parameters and metrics present inside SonarQube include:

i. **Code Smells:** Indicates potential issues in the codebase that may lead to maintainability problems or bugs.

ii. **Bugs**: Identifies actual errors or bugs in the code that need to be fixed**.**

iii. **Vulnerabilities:** Highlights security vulnerabilities in the code that could be exploited by attackers.

iv. **Code Coverage:** Measures the percentage of code covered by automated tests, helping assess the effectiveness of testing efforts.

v. **Duplication:** Identifies duplicate code blocks within the codebase, which can lead to maintenance challenges and increase technical debt.

vi. **Complexity:** Measures the complexity of code using metrics such as cyclomatic complexity, helping identify overly complex code that may be difficult to understand or maintain.

vii. **Maintainability Rating:** Provides an overall rating of the codebase's maintainability based on various factors such as code smells, duplication, and complexity**.**

viii. **Security Rating:** Provides an overall rating of the codebase's security based on the presence of vulnerabilities and security weaknesses.

ix. **Reliability Rating**: Provides an overall rating of the codebase's reliability based on factors such as bugs and code coverage.

These parameters and metrics help developers and teams assess the quality, security, and maintainability of their codebase, prioritize areas for improvement, and track progress over time.

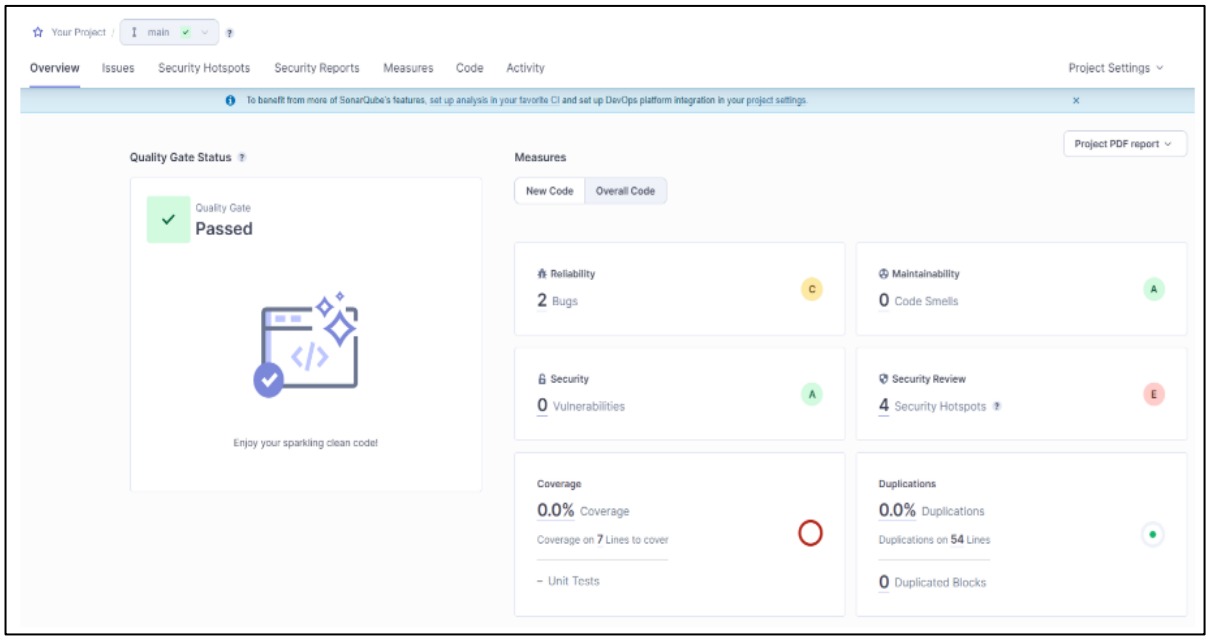


Figure 3.24 – Interface of SonarQube

3.16 SonarLint

SonarLint, an IDE extension or plugin, furnishes developers with on-the-fly feedback regarding code quality issues directly within their Integrated Development Environment (IDE). It is compatible with various IDEs such as IntelliJ IDEA, Eclipse, Visual Studio, and Visual Studio Code. SonarLint analyses code as developers write it, highlighting potential bugs, code smells, security vulnerabilities, and other issues based on rules defined by SonarQube or SonarCloud.

In contrast to SonarQube, which conducts static code analysis on the entire codebase and offers centralized reporting, SonarLint concentrates on delivering real-time feedback during the development process. This capability empowers developers to recognize and address issues as they code, thereby enhancing code quality, ensuring adherence to coding standards, and averting the introduction of issues into the codebase early in the development cycle.

In essence, SonarLint enriches the developer experience by fostering continuous improvement in code quality and ensuring consistent adherence to best practices throughout the development process.

3.17 Functions used in Licensing

In implementing licensing for features in patient monitoring systems, four key functions play pivotal roles.

1. **GetFingerprint** - The function "getFingerprint()" plays a crucial role in patient monitoring systems by retrieving unique system identifiers necessary for licensing purposes. These identifiers, which are based on system characteristics, are essential elements in creating and verifying licenses. By precisely capturing system attributes, "getFingerprint()" guarantees secure and authorized access to vital features, thereby protecting patient data and ensuring compliance with licensing regulations.

2. **InstallLicense** - In patient monitoring systems, the function "installLicense ()" assumes a critical role by integrating obtained licenses. This action authorizes access to vital features, guaranteeing compliance and security. Through the seamless integration of licenses into the system, "installLicense ()" facilitates the efficient administration of software entitlements. Consequently, healthcare providers can deliver quality care while upholding licensing agreements.

3. **Login** - In patient monitoring systems, the "login ()" function facilitates user authentication, providing access to authorized features. By verifying user credentials, it ensures secure system entry, protecting both patient data and system integrity. Through robust authentication mechanisms, "login ()" strengthens security measures, allowing healthcare providers to manage user access efficiently. Moreover, it maintains audit trails for accountability and regulatory compliance, fostering trust and confidence in the system's operation within healthcare settings.

4. **Logout** - In patient monitoring systems, the "logout ()" function securely concludes user sessions, guaranteeing data privacy and system security. By logging users out of the system after session completion, it prevents unauthorized access to patient data and sensitive information. This function bolsters data security measures, mitigating the risk of data breaches or unauthorized use. Moreover, "logout ()" preserves audit trails of user activity, fostering accountability and regulatory compliance within healthcare settings, thereby upholding the system's integrity.

Through these functions, licensing mechanisms are seamlessly integrated into patient monitoring systems, ensuring compliance, security, and controlled access to features crucial for effective healthcare delivery.

3.18 Testing of the Functions

Getfingerprint, installLicense, login and logout these are the four functions that are part of the patient monitoring systems. After the implementation, testing plays an important role so that proper validation of the functions should be taken place, and it should be reliable for the systems. For testing various Test cases were written to check the behavior of the functions that includes the successful execution, proper error handling and boundary conditions. For the getfingerprint function test case will validate that precise and accurate retrieval is done or not. The installLicense function will check that proper authorization is done or not. login and logout functions will check whether the proper session management is done or not with these testing mechanisms assurance will be maintained in the healthcare services or the providers by keeping the patient data safeguard.

Chapter – 4

Results and Discussion

4.1 GTest (GoogleTest)

Basic unit test in GoogleTest is shown below:

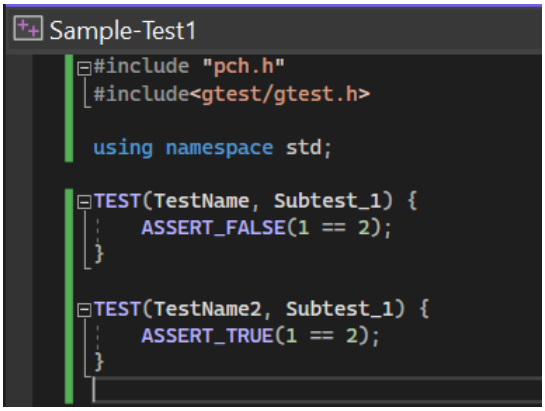


Figure 4.1 – Simple Test

```
Running main() from D:\a\_work\1\s\ThirdParty\googletest\googletest\src\gtest_main.cc
[==========] Running 2 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 1 test from TestName
[ RUN      ] TestName.Subtest_1
[       OK ] TestName.Subtest_1 (0 ms)
[----------] 1 test from TestName (1 ms total)

[----------] 1 test from TestName2
[ RUN      ] TestName2.Subtest_1
C:\Users\320236280\source\repos\Sample-Test1\Sample-Test1\test.cpp(11): error: Value of: 1 == 2
  Actual: false
Expected: true
[  FAILED  ] TestName2.Subtest_1 (1 ms)
[----------] 1 test from TestName2 (3 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 2 test cases ran. (5 ms total)
[  PASSED  ] 1 test.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] TestName2.Subtest_1

 1 FAILED TEST
```

Figure 4.2 – Output of the Sample Test

Inside this test case we have the body.

i. The first argument is the name of the test suite or test case.

ii. The second argument is the test's name within the test suite or test case.

iii. We can have 1 test case with many subtests.

iv. The assertion macros provided by GoogleTest for verifying code behavior. To use them add #include <gtest/gtest.h>.

v. testing::InitGoogleTest () method does what the name suggests, it initializes the framework and must be called before RUN_ALL_TESTS.

4.1.1 Assertion

Assert True and False are the basic assert parameters we can have many more types of assertion. In simple way assertion is the way of checking the output is true or false.

We have many asserts like ASSERT_TRUE, ASSERT_FALSE and ASSERT_EQ.

Assertion leads to 2 scenarios-

1. Success (assertion is successful)
2. Failure (assertion is fail)
    i. **Non-fatal** - The execution of code of test does not stop, it will continue after the failure.
    ii. **Fatal** - Execution of code of that test case will be stop at that place only.

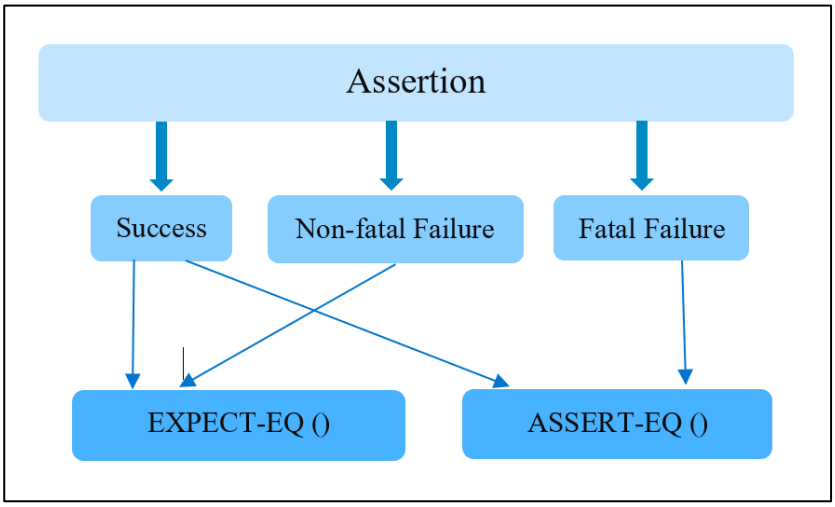


Figure 4.3 – Types of Assertion

```
#include "pch.h"
#include<gtest/gtest.h>

using namespace std;

TEST(TestName, Subtest_1) {
    ASSERT_FALSE(1 == 1);
    cout << "After Assertion" << endl;
}
```

Figure 4.4 – Basic code of Assertion

```
Running main() from D:\a\_work\1\s\ThirdParty\googletest\googletest\src\gtest_main.cc
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from TestName
[ RUN      ] TestName.Subtest_1
C:\Users\320236280\source\repos\Sample-Test1\Sample-Test1\test.cpp(7): error: Value of: 1 == 1
  Actual: true
Expected: false
[  FAILED  ] TestName.Subtest_1 (1 ms)
[----------] 1 test from TestName (2 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (4 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] TestName.Subtest_1

 1 FAILED TEST
```

Figure 4.5 - Output of the Assertion Code

Here ASSERT is false, and the Condition is true that's why ASSERT true and ASSERT false is example of fatal failure that's why it will not execute and hence we will not be able to see the output content i.e. After assertion.

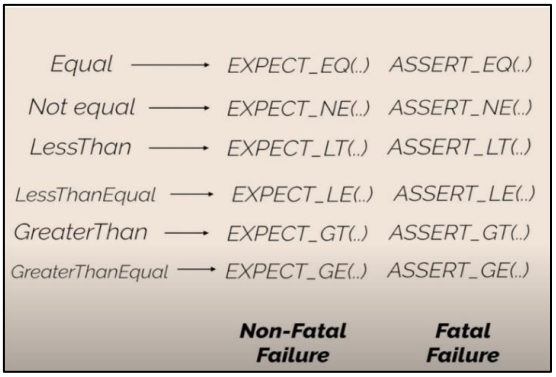


Figure 4.6 – Different types of ASSERTS and EXPECT

If we are using the ASSERT in the unit test cases with the EXPECT there are following cases:

Cases	EXPECT	ASSERT
1.	True	True
2.	True	False
3.	False	True
4.	False	False

Table 4.1 – Different cases with the ASSERT and EXPECT

```
#include "pch.h"
#include <gtest/gtest.h>

using namespace std;


TEST(TestName, subtesta_1)
{
    ASSERT_EQ(1, 1);
    EXPECT_EQ(1, 1);
    cout << "hello " << endl;


}
```

Figure 4.7 – Code for Case 1

```
Running main() from D:\a\_work\1\s\ThirdParty\googletest\googletest\src\gtest_main.cc
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from TestName
[ RUN      ] TestName.subtesta_1
hello
[       OK ] TestName.subtesta_1 (0 ms)
[----------] 1 test from TestName (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (2 ms total)
[  PASSED  ] 1 test.

C:\Users\320236280\source\repos\Sample-Test3\x64\Debug\Sample-Test3.exe (process 27592) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figure 4.8 – Output of Case 1

Note - If the position of the ASSERT and EXPECT are interchanged, then output will not change.

4.2 Unit Test Case with GoogleTest (GTest)

Any unit test must consist of 3 things –

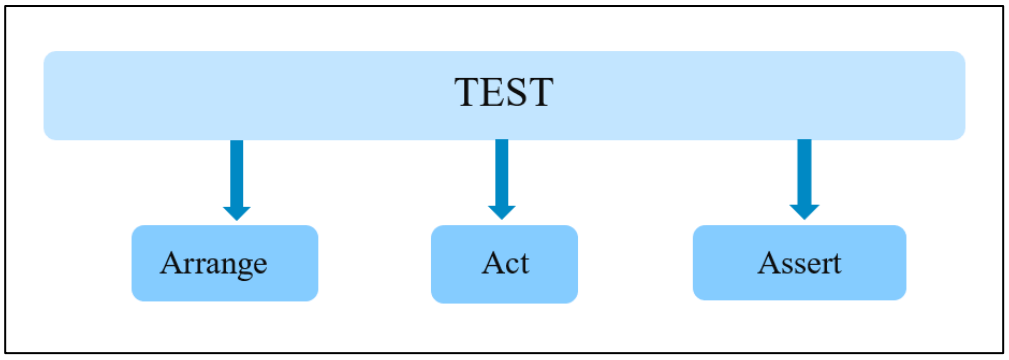


Figure 4.9 – Segregation of Test

4.2.1 Arrange - Arrange is the place where everything will be arranged.

4.2.2 Act - Act is the place where running of the test cases takes places.

4.2.3 Assert - Assert is the place where validation of the act output takes places and decide whether the assertion is successful or not.

```
#include "pch.h"
#include<gtest/gtest.h>

using namespace std;

TEST(TestName, increment_by_5) {

    //Arrange
    int value = 100;
    int increment = 5;

    //Act
    value = value + increment;

    //Assert
    ASSERT_EQ(value, 105);
}

TEST(TestName, increment_by_10) {

    //Arrange
    int value = 100;
    int increment = 10;

    //Act
    value = value + increment;

    //Assert
    ASSERT_EQ(value, 110);
}
```

Figure 4.10 – Code for Unit Test with the help of GoogleTest

```
Running main() from D:\a\_work\1\s\ThirdParty\googletest\googletest\src\gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from TestName
[ RUN      ] TestName.increment_by_5
[       OK ] TestName.increment_by_5 (0 ms)
[ RUN      ] TestName.increment_by_10
[       OK ] TestName.increment_by_10 (0 ms)
[----------] 2 tests from TestName (2 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (5 ms total)
[  PASSED  ] 2 tests.
```

Figure 4.11 – Output of above Code for Unit Test with the help of GoogleTest

There are more examples of the ASSERT and EXPECT for comparing of the strings –

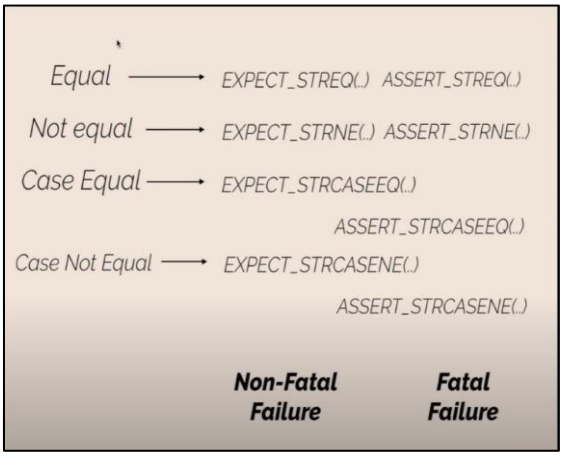


Figure 4.12 – Type of Assertion with the Strings

4.3 GMock (GoogleMock)

Mocks are used for testing the behavior of APIs/ Interfaces which will be used in component under test.

Note - mocks are used for testing the component not the APIs.

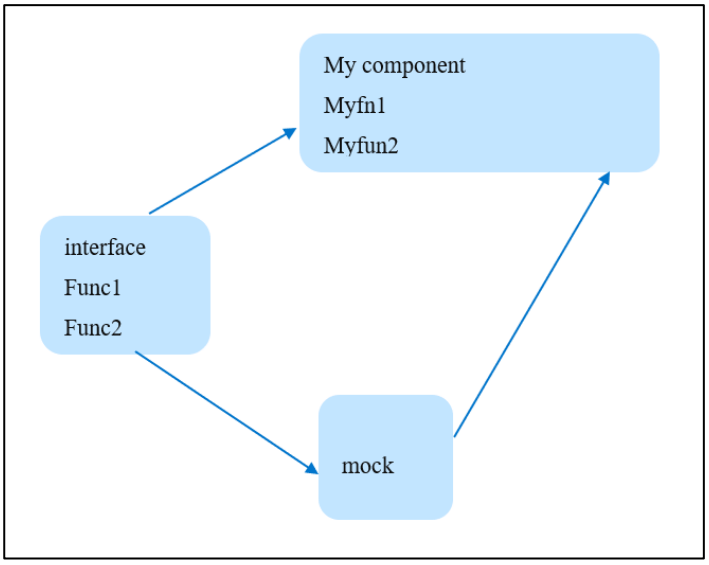


Figure 4.13 – Structure of the GMock

i. Interface was declared as an abstract class.

ii. Writing mycomponent which has 2 functions.

iii. Use interface Inside my component so there should be test for my component not for the interface but since the functionality implemented by interface may require external dependencies network connection, which is not suitable for unit testing, so write a mock for the Interface.

iv. But this mock is not testing the functionality of interface it is used for testing the functionality of mycomponent that how it will interact when it uses the interface.

v. Mocks are either stubs or fakes so mocks are used for testing the behavior Of APIs, or interfaces which will be used in the component under test and mycomponent is the name of the component which is under test and mocks are used for testing my component not the interface.

vi. Fakes look like a real implementation, but it is not the real implementation.

vii. While stubs are something which is also used alongside mock which can provide a pre cane output to a function call.

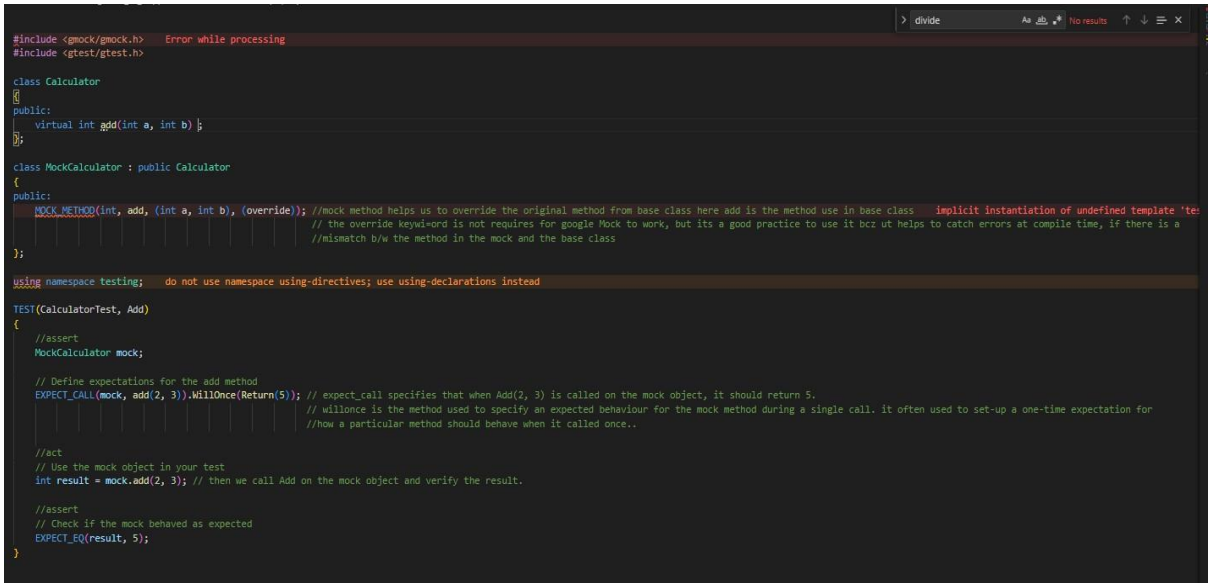


Figure 4.14 – Code of GMock

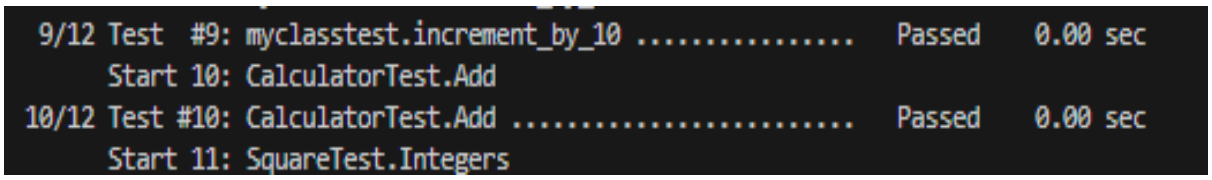


Figure 4.15 - Output of above Code of GMock

i. EXPECT_CALL (mock object, method(matchers)), the arguments of method). All matchers are defined inside the:: testing namespace.

ii. Class Calculator has the Add method i.e., under test. Create a mock for this class to isolate it from external dependencies.

4.4 Mermaid (visualization tool)

Mermaid is the visualization tool or the language through which different types of diagrams will be created such as flowcharts, sequence diagrams, Gantt charts and many more. Mermaid has that popularity among the developers instead of giving hours by creating the sequence diagrams manually mermaid makes the job much easier the only thing is that there are some set of rules that should keep in mind while creating any kind of graphs or diagrams.

Through mermaid documentations, presentations are much easier to make. Mermaids convert the text code into the fine diagrams various customization can also be done through mermaid. There is a Live

editor on which code has been written and it will develop the fine graphs and one foremost thing that it is open source anybody can access and play with it because it creates the diagrams very quickly and efficiently.

4.4.1 Sequence diagram

A Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. Mermaid can render sequence diagrams.

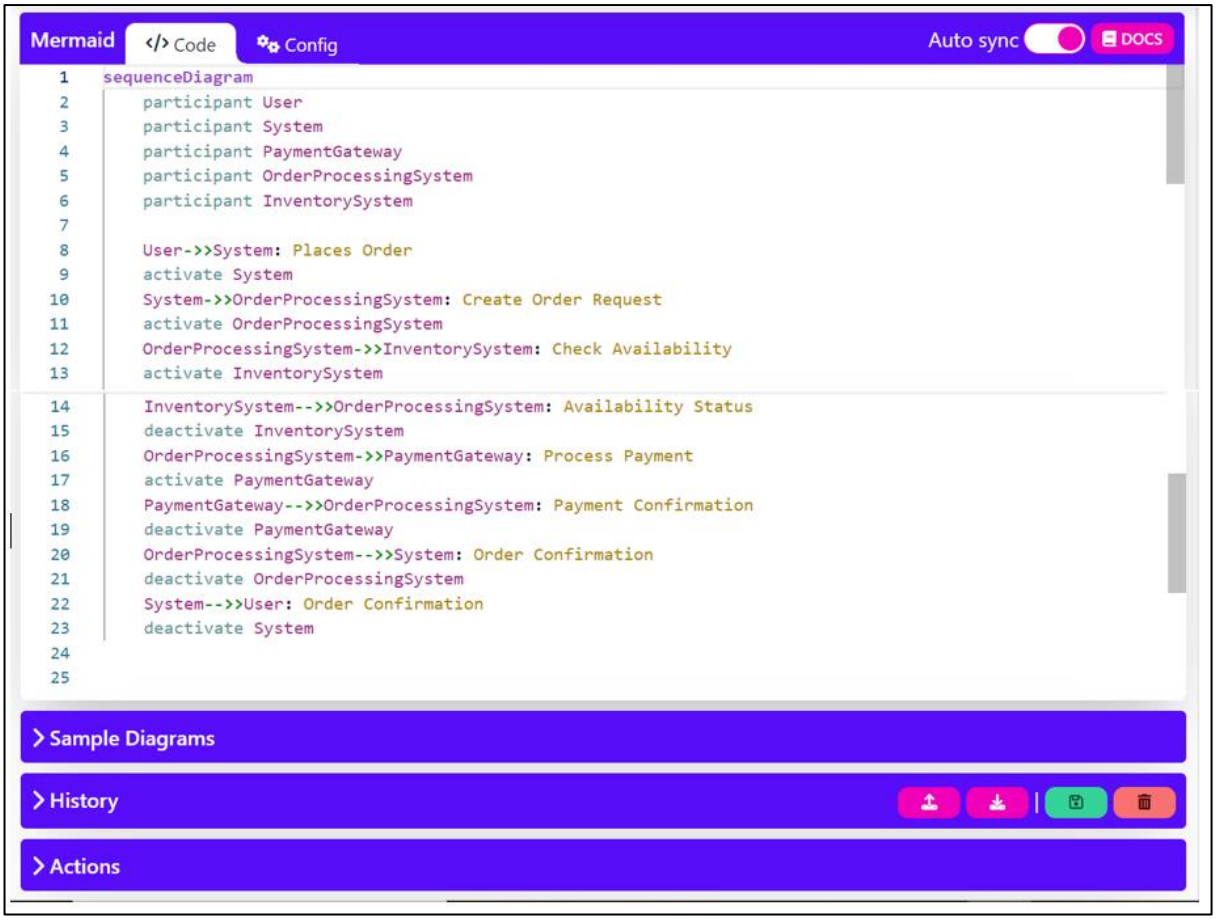


Figure 4.16 – Code for the Sequence Diagram

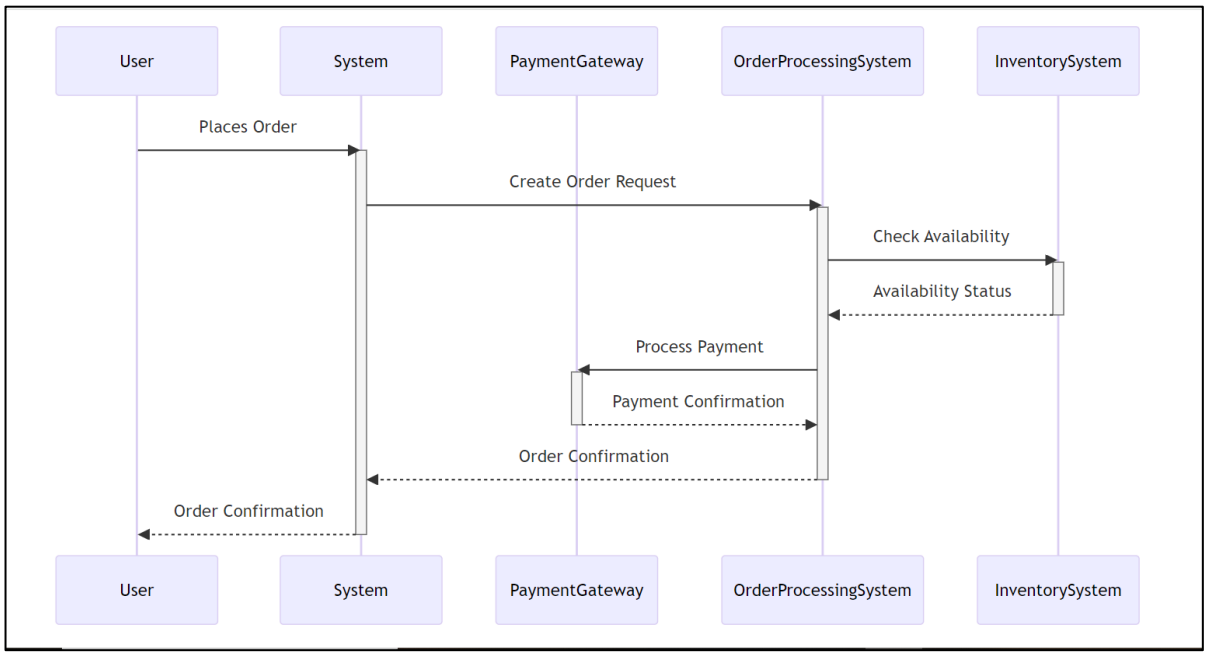


Figure 4.17 – Sequence Diagram of the above code

i. **User**: Initiates the process by placing an order.
ii. **System**: Receives the order request from the user.
iii. **OrderProcessingSystem**: Coordinates order processing.
iv. **InventorySystem**: Checks product availability.
v. **PaymentGateway**: Handles payment processing.

The sequence diagram starts with the placing an order by the user to the system. System will create an order and sent it to the order processing system after that the Inventory System will check if the order is there or not, the inventory system will sent the response to the order processing system unit with the updated status then the Order processing system will generate the functionality of the payment by the help of payment gateway after the payment is successful then the response sent back to the order processing system. Order processing system will send back the response to the system that order is confirmed, and the information is sent back to the user.

With the help of the Sequence diagram great clear visualization of the interactions within the code and functions will be easily understandable as compared to the code itself. A non-coder can easily understand the code flow by the help of the Sequence diagram how situation is being handled.

4.5 Functions of Licensing

There are four main functions which is used in this project:

i. **getfingerprint ()** - retrieves the unique system identifier, that are essential for generating and validating the licenses.

ii. **installLicense () -** install the License in the system so that authorized access should be given to the users to access the features.

iii. **login () -** login into the licensed features that should be allowed to use its functionalities.

iv. **logout ()** - the feature that was logged in should be logout or the session in which feature is being logged in should be logout so that there should be no unauthorized use.

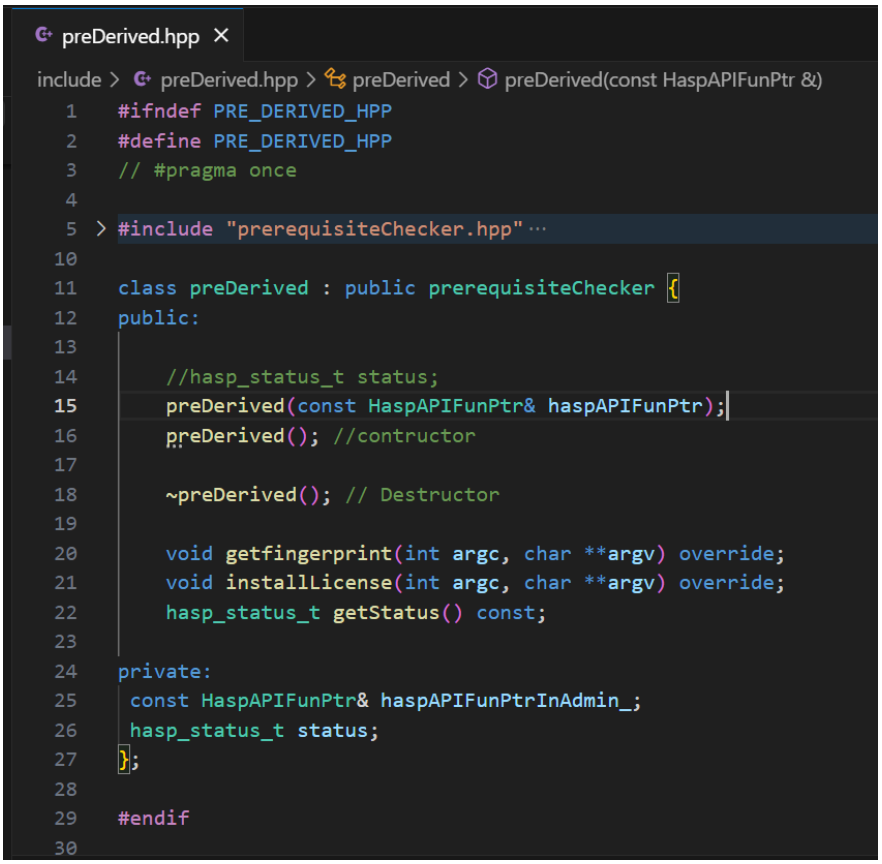


Figure 4.18 – Declaration of the getfingerprint and installLicense functions

Figure 4.19 – Definition of the getfingerprint and installLicense functions

include > sessionDerived.hpp > ...

```
#ifndef SESSIONDERIVED_HPP
#define SESSIONDERIVED_HPP

#include "sessionController.hpp"
#include "HaspAPIFuncPointer.hpp"

class sessionDerived : public sessionController {
public:
    sessionDerived();
    ~sessionDerived();
    sessionDerived(const HaspAPIFunPtr& haspAPIFunPtr);
    void login() override;
    void logout() override;
    hasp_status_t getStatus() const;

private:
    hasp_status_t status;
    const HaspAPIFunPtr& haspAPIFunPtr_;
    hasp_handle_t handle;
};

#endif

```

Figure 4.20 – Declaration of Login and Logout functions

src > sessionDerived.cpp > getStatus() const

```
#include "sessionDerived.hpp"
#include <iostream>

using namespace std;

sessionDerived::sessionDerived(const HaspAPIFunPtr& haspAPIFunPtr)
    : status(HASP_STATUS_OK), haspAPIFunPtr_(haspAPIFunPtr), handle(HASP_INVALID_HANDLE_VALUE) {}

sessionDerived::~sessionDerived() {
    // Destructor implementation
}

void sessionDerived::login() {···

void sessionDerived::logout() {···

hasp_status_t sessionDerived::getStatus() const {
    return status;
}

```

Figure 4.21 – Definition of the login and logout functions

The terminal compiles the code using the CMake build environment, resulting in an executable named "softops_practice". Executing this executable will display the output.

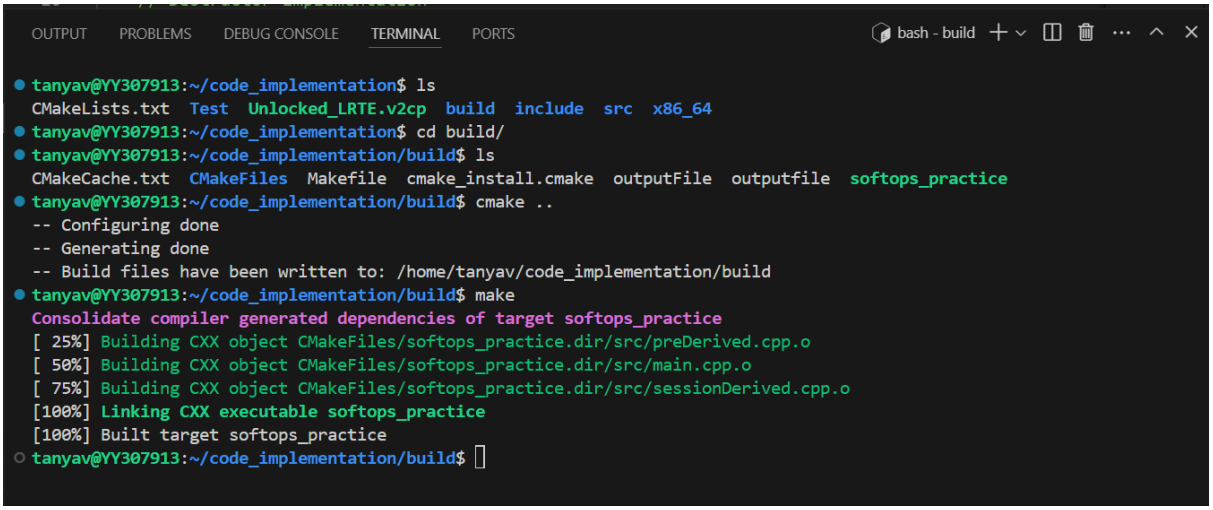


Figure 4.22 – running of cmake command

4.6 Testing of the functions

Testing means checking the behavior of the functions in all aspects of how they behave, testing covers all the scenarios like valid inputs, non-valid inputs, edge cases. Test cases are implemented to verify the functionality of the functions, its performance, and reliability checks whether the tested functions behave as they written or not.

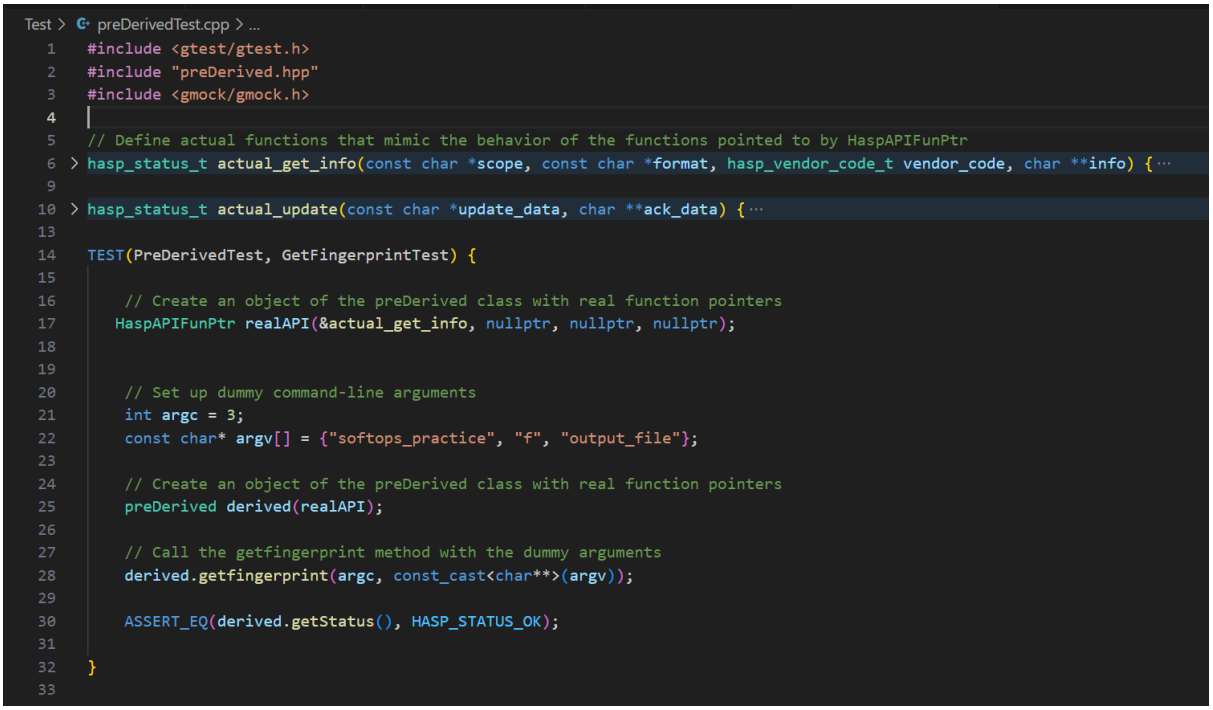


Figure 4.23 – Testing of the getfingerprint function

Test > preDerivedTest.cpp > ...

```
TEST(PreDerivedTest, InstallLicenseTest) {
    std::string fileData = "License Update Info";
    std::string filePath = "Unlocked_LRTE.v2cp"; // Specify the file name

    // Open the file specified by the file path for writing
    std::ofstream licenseFile(filePath);
    if (!licenseFile.is_open()) {
        std::cerr << "Could not open file " << filePath << std::endl;
        FAIL(); // Fail the test if unable to open the file
    }

    // Write the content of the fileData string to the file opened by licenseFile
    licenseFile << fileData;
    licenseFile.close(); // Close the file

    HaspAPIFunPtr realAPI(nullptr, &actual_update, nullptr, nullptr);
    preDerived derived(realAPI);

    int argc = 3;
    const char* argv[] = {"softops_practice", "f", filePath.c_str()}; // Pass the file path

    derived.installLicense(argc, const_cast<char**>(argv));

    // Remove the file created for testing purposes
    remove(filePath.c_str());

    ASSERT_EQ(derived.getStatus(), HASP_STATUS_OK);
}
```

Figure 4.24 – Testing of the installLicense function

Test > sessionDerivedTest.cpp > ...

```
#include <gtest/gtest.h>
#include "sessionDerived.hpp"
#include <gmock/gmock.h>
//#include <sstream>

> hasp_status_t actual_login(hasp_feature_t feature_id, hasp_vendor_code_t vendor_code, hasp_handle_t *handle) {···

> hasp_status_t actual_logout(hasp_handle_t handle) {···

// Test case for login method without mocking
TEST(SessionDerivedTest, LoginTest) {

    // Create an object of the sessionDerived class with real function pointers
    HaspAPIFunPtr realAPI(nullptr, nullptr, &actual_login, nullptr);
    sessionDerived session(realAPI);

    // Call the login method
    session.login();

    ASSERT_EQ(session.getStatus(), HASP_STATUS_OK);
}
```

Figure 4.25 – Testing of login function

```
30  // Test case for logout method without mocking
31  TEST(SessionDerivedTest, LogoutTest) {
32
33      HaspAPIFunPtr realAPI(nullptr, nullptr, nullptr, &actual_logout);
34      sessionDerived session(realAPI);
35
36      session.logout();
37
38      ASSERT_EQ(session.getStatus(), HASP_STATUS_OK);
39  }
```

Figure 4.26 – Testing of logout function

Upon compiling the code using the CMake build environment, the terminal produces an executable named "test_project". Executing this executable showcases the output. To conduct test cases, subfolders named "Test" are created within the main folder, "code_implementation". Within this folder, test cases are authored to validate the functions' functionality. The "Test" subfolder enhances clarity and organization between the main codebase and its corresponding test cases.

```
OUTPUT  PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS                    bash - build
tanyav@YY307913:~/code_implementation$ ls
CMakeLists.txt  Test  Unlocked_LRTE.v2cp  build  include  src  x86_64
tanyav@YY307913:~/code_implementation$ cd Test/
tanyav@YY307913:~/code_implementation/Test$ ls
CMakeLists.txt  build  main.cpp  preDerivedTest.cpp  sessionDerivedTest.cpp
tanyav@YY307913:~/code_implementation/Test$ cd build/
tanyav@YY307913:~/code_implementation/Test/build$ cmake ..
-- Configuring done
-- Generating done
-- Build files have been written to: /home/tanyav/code_implementation/Test/build
tanyav@YY307913:~/code_implementation/Test/build$ make
Consolidate compiler generated dependencies of target test_project
[ 20%] Building CXX object CMakeFiles/test_project.dir/preDerivedTest.cpp.o
[ 40%] Building CXX object CMakeFiles/test_project.dir/sessionDerivedTest.cpp.o
[ 60%] Building CXX object CMakeFiles/test_project.dir/home/tanyav/code_implementation/src/preDerived.cpp.o
[ 80%] Building CXX object CMakeFiles/test_project.dir/home/tanyav/code_implementation/src/sessionDerived.cpp.o
[100%] Linking CXX executable test_project
[100%] Built target test_project
tanyav@YY307913:~/code_implementation/Test/build$ ls
 CMakeCache.txt   Makefile              cmake_install.cmake   output_file   'test_project[1]_include.cmake'
 CMakeFiles       UnlockedLicense.v2cp  file.c2v              test_project  'test_project[1]_tests.cmake'
```

Figure 4.27 – Terminal for running the Test cases

When running the "test_project" executable, one encounters the output.

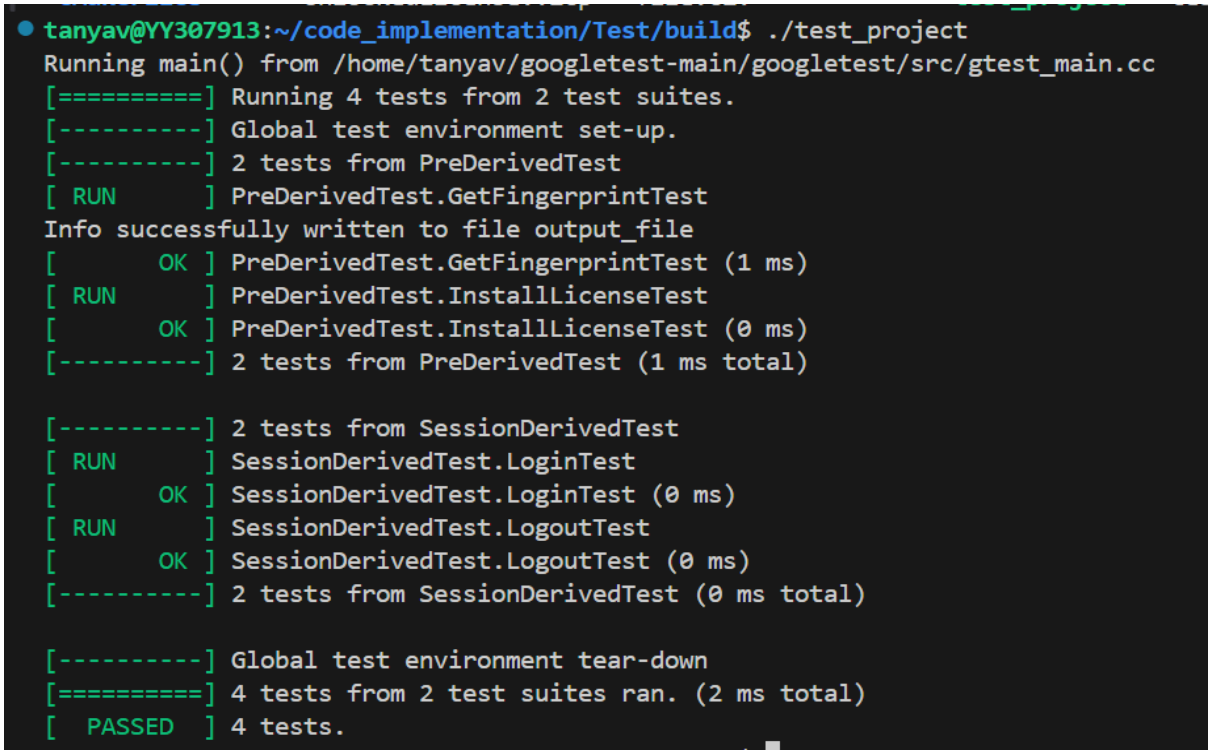


Figure 4.28 – Successfully all Test cases were passed

Chapter 5
Concluding Remarks and Future Scope

5.1 Concluding Remarks

As the conclusion of this Thesis, Licensing plays an important role in the healthcare industry which guarantees the highest safety, quality, and innovation for the patient monitoring systems. Licensing assures that it should develop trust for the patient's well-being. Quality assures that during the measurement of the patient data it should be as much as accurate and precise because if the value is differ by one single point then patient heath should be in danger and proper treatment will not be take place, so various experiments will take place to satisfy the values related to patient in the healthcare systems. Regularly updating in the systems will help to significantly increase the effectiveness in the patient care. Licensing not only encourages innovation but also helps in the patient's safety rights because of the safety issues only various license models comes into picture so that licensing could be able to reduce the risk from the patient health. Licensing is not only for the patient health care but also plays an important role in the business of any industry or the organization through licensing great hike in business also happens that's why the users from the different organizations purchase the license and use it according to their needs or purpose so in the last licensing give the trust and safety not only for the doctors but also for the patient.

5.2 Future Scope

In the future, licensing is already in the field of the software but in the future it is also the part of hardware as well because licensing wins the trust among the people because through licensing users not using any pirated software for their systems they were using the trusted and legal software among them. Licensing now in some of the heath care industries but in future every heath care industry will have this feature between them. The heath care institutions will depend on the licensed features rather than the non-licensed feature because licensing will take the patient care and health more sincerely rather than taking the patient health for granted why because through licensing the tools or systems will not be pirated anymore, and it will maintain the quality standards as well. Each function during the process of licensing will undergo so many testing phases so that there will be no such scenarios come into picture later which cannot be handle by the developers.

Healthcare industries will adopt the licensed feature because they all are tested before coming into the running state or the execution state these licensed features will drastically impact on the patient care and speedy recovery because the parameters related to the patient heath will calculate more accurately so proper treatment will be provided to the patients. Licensing also promises the security for the patient sensitive data by keeping the quality first throughout the development and testing that's why it will improve the efficiency, accuracy and safety in the patient treatment with the successful implementation new era of innovation will flow in the health care industries.

References

[1] Josh Lerner and Jean Tirole, “THE SIMPLE ECONOMICS OF OPEN SOURCE”,2005.

[2] "A Survey of Intellectual Property Licensing Practices in High-Technology Industries" by Levin et al.

[3] "Licensing in a Market for Technology" by Arora, Fosfuri, and Rende.”.

[4] Halina Kaminski and Mark Perry, "A Pattern Language of Software Licensing", EuroPLoP' 2005, Tenth European Conference on Pattern Languages of Programs, Irsee, Germany.

[5] April 2021 web server survey. [Online]. Available: https: //news.netcraft.com/archives/category/web-server-survey/

[6] (2016) Open source won. so, now what? [Online]. Avalable : https : //www.wired.com/2016/08/open-source-won-now/ ? GuidesLearnMore/

[7] G. von Krogh, S. Spaeth, and S. Haefliger, “Knowledge reuse in open-source software: An exploratory study of 15 open-source projects,” in Proceedings of the 38th Annual Hawaii International Conference on System Sciences, 2005, pp. 198b–198b.

[8] Y. Golubev, M. Eliseeva, N. Povarov, and T. Bryksin, “A study of potential code borrowing and license violations in java projects on GitHub,” in Proceedings of the 17th International Conference on Mining Software Repositories, 2020, pp. 54–64.

[9] D. M. German, M. Di Penta, and J. Davies, “Understanding and auditing the licensing of open-source software distributions,” in 2010 IEEE 18th International Conference on Program Comprehension. IEEE, 2010, pp. 84–93.

[10] S. Romansky, C. Chen, B. Malhotra, and A. Hindle, “Sourcerer’s apprentice and the study of code snippet migration,” ArXiv, vol. abs/1808.00106, 2018.

[11] C. Vendome, M. Linares-Vasquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk, "License Usage and Changes: A Large-Scale Study of Java Projects on GitHub," in 2015 IEEE 23rd International Conference on Program Comprehension, Florence, 2015, pp. 218-228.

[12] Daniel A. Almedia, Gail C. Murphy, Greg Wilson, Michael Hoye, “Investing whether and how software developers understand open-source software licensing,” Empirical Software Engineering. Vol.24, pp. 211-239, 2019.

[13] G. M. Kapitsaki, F. Kramer, “Open-source license violation check for spdx files,” in International Conference on Software Reuse, 2015, pp. 90-105.

[14] G. Kapitsaki, G. Charalambous, “Modeling and recommending open-source licenses with findOSSLicense,” IEEE Transactions on Software Engineering, 2019.

[15] Hithru De Alwis, Adeesha Wijayasiri, Shamila De Silva, and Kasun De Silva, "BLOCKCHAIN-BASED SOFTWARE SUBSCRIPTION AND LICENSES MANAGEMENT SYSTEM", 2023 8th International Conference on Information Technology Research (ICITR) | 979-8-3503-5950-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICITR61062.2023.10382813.

[16] A. Goebel, The principle of exhaustion and the resale of downloaded software – The UsedSoft/Oracle case,” Eur. Law Report. (ELR), no. 9, 2012.

[17] T. Vinje, V. Marsland, and A. Gartner, “Software Licensing After Oracle v. UsedSoft,” Comput. Law Rev. Int., vol. 13, no. 4, pp. 97-102, 2012.

[18] Mika Murtojärvi and Mika Johnsson, “Determining the Proper Number of Software Licenses”, IEEE Transactions on Pattern Analysis and Machine Intelligence 33:305-315.

[20] Bjarne Stroustrup,” C++ Programming: Implementation of the Licensing System for a Software Product”.

[21] J. Souli, “C ++ Language Tutorial,” Cogn. Technol. Work, 2007, doi: 10.1007/978-3-540-74444-3.

[22] David Abrahams and Aleksey Gurtovoy “C++ Template Metaprogramming”

[23] Manikas, K. Revisiting software ecosystems Research: A longitudinal literature study. Journal of Systems and Software, 117, 84–103. doi: 10.1016/j.jss.2016.02.003.

[24] Manikas, K., & Hansen, K. Software ecosystems – A systematic literature review. Journal of Systems and Software, 86(5), 1294–1306. doi: 10.1016/j.jss.2012.12.026 IBM. (2016). Innovation in the API economy: Building winning experiences and new capabilities to compete.

[25] Niu, H., Keivanloo, I., & Zou, Y. (2016). API usage pattern recommendation for software development. Journal of Systems and Software, 1–13.

[26] Qiu, D., Li, B., & Leung, H. (2016). Understanding the API usage in Java. Information and Software Technology, 73, 81–100. doi: 10.1016/j.infsof.2016.01.011

[27] Karuturi Sneha and Malle Gowda M, "Research on Software Testing Techniques and Software Automation Testing Tools",International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS-2017)

[28] R. Chauhan and I. Singh, "Latest Research and Development on Software Testing Techniques and Tools", INPRESSCO, International Journal of Current Engineering and Technology.

[29] S. P and D. N, "Automation of Software Testing in Agile Development – An Approach and Challenges with Distributed Database Systems", GJRA - GLOBAL JOURNAL FOR RESEARCH ANALYSIS, vol. 3, no. 7, 2014.

[30] S. Sharma, "Study and Analysis of Automation Testing Techniques", Journal of Global Research in Computer Science, vol. 3, no. 12, 2012.

[31] "Analysis of Automation and Manual Testing Using Software Testing Tool", IJIACS, vol. 4, ISSN 2347 – 8616, 2017.

[32] S. Thummalapenta, S. Sinha and N. Singhania, "Automating Test Automation", IBM T. J. Watson Research Center, IBM Research - India, 2017.

[33] "An Approach of Software Design Testing Based on UML Diagrams", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, no. 2, 2014.

[34] "A Unique Technique to Handle Complexity and Improve the Effectiveness of Test Cases in Software Testing", Journal of Innovative Technology and Education, vol. 2, 2015.

[35] N. Petrović +, M. Radenković +, S. Cvetković ++, and D. Rančić +,"Model-driven Automated gMock Test Generation for Automotive Software Industry “, XV International SAUM Conference on Systems, Automatic Control and Measurements Niš, Serbia, September 09th - 10th, 2021

[36] What Is Google C++ Mocking Framework? [Online]. Avalable on: https://chromium.googlesource.com/external/github.com/google/goog latest/+/refs/tags/release-1.8.0/googlemock/docs/ForDummies.md

[37] J. Langer, Modern C++ Programming with Test-Driven Development, The Pragmatic Programmers, 2013.

[38] GoogleTest [online]. Available on: https://github.com/google/googletest

[39] GMock Cookbook [online]. Available on: http://google.github.io/googletest/gmock_cook_book.html

[40] Copyright@ Philips Innovation Campus, India.

M.Tech. Thesis 2024

ORIGINALITY REPORT

10%	7%	3%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Asia Pacific International College Student Paper	1%
2	jojozhuang.github.io Internet Source	1%
3	zero.sci-hub.se Internet Source	1%
4	Zhiyou Liu, Zili Zhang, Zhiqiang Wang, Jing Peng, Sheng Wu. "Choosing an Open Source License Based on Software Dependencies", 2021 IEEE International Conference on Software Engineering and Artificial Intelligence (SEAI), 2021 Publication	<1%
5	mermaid.js.org Internet Source	<1%
6	unpkg.com Internet Source	<1%
7	www.security-science.com Internet Source	<1%

8	apriorit.com Internet Source	<1 %
9	Submitted to University of Denver Student Paper	<1 %
10	www.researchgate.net Internet Source	<1 %
11	www.nuvovis.com Internet Source	<1 %
12	Submitted to Eiffel Corporation Student Paper	<1 %
13	arxiv.org Internet Source	<1 %
14	resources.docs.salesforce.com Internet Source	<1 %
15	www.geeksforgeeks.org Internet Source	<1 %
16	Submitted to Monash University Student Paper	<1 %
17	documentation.dnanexus.com Internet Source	<1 %
18	forge.chapril.org Internet Source	<1 %
19	Karuturi Sneha, Gowda M Malle. "Research on software testing techniques and software	<1 %

automation testing tools", 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), 2017
Publication

20	www.ibm.com Internet Source	<1%
21	Lentin Joseph. "Robot Operating System for Absolute Beginners", Springer Science and Business Media LLC, 2018 Publication	<1%
22	google.github.io Internet Source	<1%
23	Mauro Femminella, Matteo Pergolesi, Gianluca Reali. "Performance Evaluation of Edge Cloud Computing System for Big Data Applications", 2016 5th IEEE International Conference on Cloud Networking (Cloudnet), 2016 Publication	<1%
24	Submitted to University of North Texas Student Paper	<1%
25	www.grin.com Internet Source	<1%
26	Submitted to National College of Ireland Student Paper	<1%
27	Submitted to University of South Australia Student Paper	

		<1%
28	Submitted to Southern New Hampshire University - Continuing Education Student Paper	<1%
29	Submitted to University of Mauritius Student Paper	<1%
30	cmake.org Internet Source	<1%
31	www.howtogeek.com Internet Source	<1%
32	Submitted to Adtalem Global Education Student Paper	<1%
33	Submitted to Higher Education Commission Pakistan Student Paper	<1%
34	Hithru De Alwis, Adeesha Wijayasiri, Shamila De Silva, Kasun De Silva. "Blockchain-Based Software Subscription and Licenses Management System", 2023 8th International Conference on Information Technology Research (ICITR), 2023 Publication	<1%
35	Submitted to Oregon State University Student Paper	<1%

36	Submitted to University of Cincinnati Student Paper	<1%
37	"EU Internet Law in the Digital Single Market", Springer Science and Business Media LLC, 2021 Publication	<1%
38	Alexey Lastovetsky. "Parallel Computing on Heterogeneous Networks", Wiley, 2003 Publication	<1%
39	es.slideshare.net Internet Source	<1%
40	s-space.snu.ac.kr Internet Source	<1%
41	www.browserstack.com Internet Source	<1%
42	www.harness.io Internet Source	<1%
43	Submitted to University of New South Wales Student Paper	<1%
44	knsv.github.io Internet Source	<1%
45	shiftsync.tricentis.com Internet Source	<1%
46	5dok.net Internet Source	<1%

47	Hyung-Min Koo, In-Young Ko. "Construction and utilization of problem-solving knowledge in open source software environments", Journal of Systems and Software, 2017 Publication	<1 %
48	Swain, Da. "The role of the dynamics of relative motion in information passing in natural and engineered collective motion", Proquest, 2014. Publication	<1 %
49	www.iseb-istqbsoftwaretesting.co.uk Internet Source	<1 %
50	"Pervasive Computing Technologies for Healthcare", Springer Science and Business Media LLC, 2024 Publication	<1 %
51	"Service-Oriented Computing", Springer Science and Business Media LLC, 2017 Publication	<1 %
52	Submitted to East Tennessee State University Student Paper	<1 %
53	Lieyu Lv, Ling Xiong, Fagen Li. "PCPHE: A privacy comparison protocol for vulnerability detection based on homomorphic encryption", Journal of Information Security and Applications, 2024 Publication	<1 %

54	cps-vo.org Internet Source	<1 %
55	ebin.pub Internet Source	<1 %
56	fossies.org Internet Source	<1 %
57	ijarcce.com Internet Source	<1 %
58	opus.lib.uts.edu.au Internet Source	<1 %
59	stackoverflow.com Internet Source	<1 %
60	thesai.org Internet Source	<1 %
61	trepo.tuni.fi Internet Source	<1 %
62	ws001.sspa.juntadeandalucia.es Internet Source	<1 %
63	www.igi-global.com Internet Source	<1 %
64	www.ijariit.com Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		