**iCare-Healthcare Information System**

**SOEN 6471 - Advanced Software Architectures**

**Professor: Dr.Malleswara Rao Talla**

Hebbar Nandakishore 27644396, KeerthanaGudavalli27588569,

Leili Jahanshahi Monfared 27724810, Ozioma Aniagu 27667280,

Parinaz Barakhshan 27675518, Ramandeep Kaur Mehare27664613,

Reza Shalchian 27869703, Salameh Abdollahi Sohi 27768583,

Vijay Shah 27735146

*Abstract*— this document involves the study and evaluation of our advanced software architecture course project, aiming to design, describe and evaluate an architecture of a Health Information System (HIS) based on the standards of the health informatics discipline. Basically, this acts as a bridge between patients and health facilities. The health information system collects data from the health sector and other relevant sectors. This visualization of treatment history helps in giving better services to patients.

Keywords - Health Information System, Electronic Health Record, Stakeholder, Architectural Views & Patterns.

# **INTRODUCTION**

Strong Health Systems are important to achieving better health outcomes. A properly functional HIS (Health Information System) delivers adequate services to the right people at the right time. This enables policymakers, managers, and individual service providers to make proper decisions about patient care and even national budgets. A robust HIS is the backbone of Health Services which support greater transparency and accountability by increasing accessibility of health information. Unfortunately, many low and middle-income countries (LMICs) have a long way to go to achieve these goals [1].

The iCare system can be useful for countries even with different level of capacities in four major aspects in mind: collation of medical data, data analyses, improved capacity to manage HIS, addressing information gaps with strong methods, tools and rigorous evaluation.

The HIS provides the underpinnings for decision-making in medical treatments. The Health Information System collects data from the health sector and other relevant sectors. The system analyses the data and ensures its overall quality, relevance and timeliness. It also converts data into information for health-related decision-making [2].

These are properly monitored, timely evaluated, managed and transmitted when required. In general, this acts as a bridge between patients and health services.

An Electronic Health Record (EHR) is in digital format, consists of all the various information created for the patients from various medical fields such as radiology, pathology and patient health record. It may include medical record, immunization status, test results, radiology images, CT scans, vital signs and personal information like age and weight. This format can be shared among doctors and patients. This visualization of treatment history helps in getting status on the services to patients. iCare is a type of HIS that manages and monitors medical history, prescriptions, operations and lab tests of patients in a secure way. It also contains features such as drug database, symptom checklist, scheduling appointment for patients and generating notifications. All these features help doctors to give prescriptions to patients online and also in person.

# **II. SYSTEM STAKEHOLDERS IDENTIFICATION**

According to the Standard, Stakeholders are those individuals, teams or organizations that have interest in the system [3].The following stakeholders were considered when preparing System Architectural Description (SAD) of iCare system. The stakeholders are grouped into users, operators, acquirers, owners, suppliers, developers, builders and maintainers [4].

S1: **Project Manager**

**Group**: Owner

**Description**: This stakeholder supervises, monitors and allocate resources for the development of the system.

**Responsibility**: Management and balancing of cost, time, scope and quality in the project.

S2: **Architect**

**Group**: Builder

**Description**: An architect develops and documents the iCare system architecture.

**Responsibility**: Negotiating with stakeholders and project manager to ensure the project success.

S3: **Maintainer**

Group: Maintainer

**Description**: A software maintainer who is fixing bugs found during testing of the system.

**Responsibility**: Upgrading and modification of the system when need be.

S4: **Tester**

**Group**: Developer

**Description**: Tests the system to ensure that it is suitable for use.

**Responsibility**: Assures that system environment provided for the software is sufficient.

S5: **System Analyst**

**Group**: Developer

**Description**: System Analyst analyzes the architecture to make sure it meets critical quality attribute requirements.

**Responsibility**: Ensures that the system adapt to existing systems and integrates new features or improvements.

S6: **Designer**

**Group**: Developer

**Description**: System Designer draws up detailed design documentation including charts and diagrams that indicate the various components involved in the system.

**Responsibility**: Works with analysts on the feasibility of a conceptual design by taking technical specifications prepared by the analyst and designing system components to meet the system requirements.

S7: **User**

**Group**: User

**Description**: This is the end user of the iCare system. **Responsibility**: The user checks whether the functionalities of the system are being delivered.

S8: **Database Administrator**

Group: Operator

**Description**: Database Administrator maintains the staff record.

**Responsibility**: Database Administrator adds, removes and modifies the system database.

S9: **Health Care Specialist**

**Group**: Acquirer

**Description**: Health Care Specialist has knowledge of medical science.

Responsibility: Health Care Specialist inputs medical treatment decisions into the system.

S10: **Hacker**

**Group**: User

**Description**: Hacker is a negative stakeholder.

**Responsibility**: Hacker wants to modify records without permission

**STAKEHOLDERS CONCERNS**

The stakeholder concerns are vital to Software Architecture (SA). Each concern of iCare system is held by one or many stakeholders (minimum one) and include the following: portability, modifiability, availability, security, performance, testability, usability, interoperability, and user affordability. The table shows the most relevant concerns and related quality attributes of the iCare system.

**C 01**:

Concern: Is the system adaptable to new environment?

**Quality Attribute:** Portability

**C 02:**

**Concern:** Can the system be updated to a new version?

**Quality Attribute:** Modifiability

**C 03:**

**Concern:** Does the system protect patient information from the unauthorized persons?

**Quality Attribute**: Security

**C 04:**

**Concern**: Is the system affordable?

**Quality Attribute**: Affordability

**C 05:**

**Concern:** Is data modified by appropriate user?

**Quality Attribute:** security

**C 06:**

**Concern:** Does the system give quick response?

**Quality Attribute:** Performance

**C 07:**

**Concern:** Does system recover fast from faults?

**Quality Attribute:** Availability

**C 08:**

**Concern:** Are all the features of system easy to test?

**Quality Attribute:** Testability

**C 09:**

**Concern:** Is it easy for any user to operate the system?

**Quality Attribute:** Usability

**C 10:**

**Concern:** Are features of system able to interact among themselves?

**Quality Attribute:** Interoperability

# **USE CASE VIEW**

### **Purpose:** Use cases are pertinent in terms of illustrating information based on high level system requirements and actors defining system architecture. Use cases help to define system boundaries, capabilities and flow of information (using relations such as includes etc) while providing adequate abstraction to implementation details of the system. Moreover, this view provides all the main system functionalities from the user point of view. This view is extremely useful in communicating system architecture to non-technical stakeholders and extracting relevant feedback during early stage of the project.

### **Stakeholders:** The stakeholders for use case view of the system architecture are Users, Administrator, Maintainer, Tester, Regulator/Government.

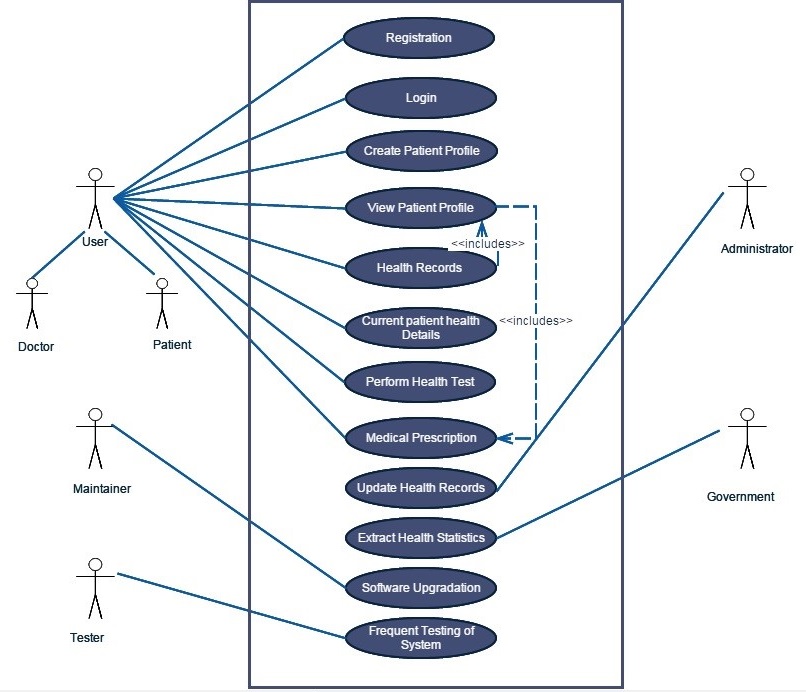


Fig.1. Use Case Diagram

|  |  |
| --- | --- |
| **Use Case Name** | Registration |
| **Use Case ID** | UC-1 |
| **Actors** | User (Patient, Doctors), System Administrator |
| **Description** | The person who is not registered to the system can easily create the account then access the services. The system does not allow user to create his/her own health record due to concern of legitimacy. While creating patient record, the following information is captured: Patient demographics, contact information, reason for visiting, etc. |

|  |  |
| --- | --- |
| **Use Case Name** | Login |
| **Use Case ID** | UC-2 |
| **Actors** | User (Patient, Doctors), System Administrator |
| **Description** | The registered user can easily accesses the system and check the previous records. This will help to keep the records in the archive but not available for manipulation. |

|  |  |
| --- | --- |
| **Use Case Name** | Create Patient Profile |
| **Use Case ID** | UC-3 |
| **Actors** | User (Patient, Doctors), System Administrator |
| **Description** | The system does not allow user to create his/her own health records due to concern of validity. While creating patient record, the following information is captured: Patient demographics, contact information, reason for visiting etc. |

|  |  |
| --- | --- |
| **Use Case Name** | View Patient Profile |
| **Use Case ID** | UC-4 |
| **Actors** | User (Patient, Doctors) |
| **Description** | Doctors view records of only patients who are undergoing treatment. Patients can also look into their own medical records but not in other patients’ record. |

|  |  |
| --- | --- |
| **Use Case Name** | Health Records |
| **Use Case ID** | UC-5 |
| **Actors** | User (Patient, Doctors), System Administrator |
| **Description** | Doctor can update patient health records while going through treatment under his/her supervision. The record consist information about the disease, treatment, prescription and history of health checks. |

|  |  |
| --- | --- |
| **Use Case Name** | Current Health Details |
| **Use Case ID** | UC-6 |
| **Actors** | User(Patient, Doctor) |
| **Description** | Patient can update regular health indicators such as blood sugar level, blood pressure and body temperature into the system using mobile/Web application to be used with doctors. This also has ability to let emergency staff to tackle critical first assistance without delay. |

|  |  |
| --- | --- |
| **Use Case Name** | Perform Health Test |
| **Use Case ID** | UC-7 |
| **Actors** | User (Patient, Doctor) |
| **Description** | Patient can perform the basic tests like blood sugar level, blood pressure and body temperature measured into the system through mobile/Web application for doctors. |

|  |  |
| --- | --- |
| **Use Case Name** | Medical Prescription |
| **Use Case ID** | UC-8 |
| **Actors** | User (Patient, Doctor) |
| **Description** | Doctors can only prescribe drugs. Dosage is defined so that pharmacist can administer appropriate care to the patient. Doctors can change the drug prescription after looking at patient’s conditions. |

|  |  |
| --- | --- |
| **Use Case Name** | Update Health Records |
| **Use Case ID** | UC-9 |
| **Actors** | User (Patient, Doctors), System Administrator |
| **Description** | A patient health record can consist of attached health reports such as X-rays, ECG reports etc. The system stores each report in an electronic format to keep records of test performed for certain disease. Doctor can update patient health records while undergoing treatment under his/her supervision. |

|  |  |
| --- | --- |
| **Use Case Name** | Extract Health Statistics |
| **Use Case ID** | UC-10 |
| **Actors** | Regulatory, Doctors |
| **Description** | This the most important use for an intelligent system. The system can provide reports on patients based on age, gender, disease to regulators to make budget decisions. Also, such system can alert doctor about a patient who is not responding to treatment based on the unusual medical health indicators in record. |

|  |  |
| --- | --- |
| **Use Case Name** | Software Upgrade |
| **Use Case ID** | UC-11 |
| **Actors** | Maintainer |
| **Description** | Maintainer modifies the system software to improve performance or other attributes. |

|  |  |
| --- | --- |
| **Use Case Name** | Frequent Testing of the System |
| **Use Case ID** | UC-12 |
| **Actors** | Tester |
| **Description** | The Tester tests that whether the system is working properly. Frequent Testing is an essential to make sure the system is working properly. |

# **CLASS DIAGRAM**

The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.

Problem domain items are modeled based on the stakeholder’s basic requirements. The most significant elements in the class diagram are the following:

* **User**: It defines the basic end-user of the system, along with an inheritance hierarchy that allows the system to represent different users with different responsibilities in the system (e.g. Patient, Doctor, and System Administrator). Each user has an Id as

a unique identifier.

* **Schedule**: Defines a time interval entity, which could be used either to represent the Health Professional availability, as well as to help define an Appointment.
* **Health Record**: The main entity of interest regarding the Patient. Apart from informing patient health status (diseases, medications taken, tests results and health measurements/indicators) – that is the main source of information for the hospital staff to treat the patient – it is also useful in building statistics.
* **Fee**: Defines a fee that must be charged to the patient.
* **Prescription**: Part of the Patient’s Health Record, the Prescription is the main source of information to the Pharmacist, as well as important item for the other health professionals.
* **Health Measurements**: The measurements of a particular health vitals such as Blood pressure, pulse rate, temperature and general pathological test data can be captured and sent for the doctor’s review.
* **Administrator**: It has three generalized classes**;** System Administrator, Database Administrator and

Security Administrator. Security Administrator handles some functionalities of the system such login to the system, checking the security of the system.

* **Health professional**: Doctor is extended (generalized) from this class and all the health professional and other administrators are generalized from the class administrator.

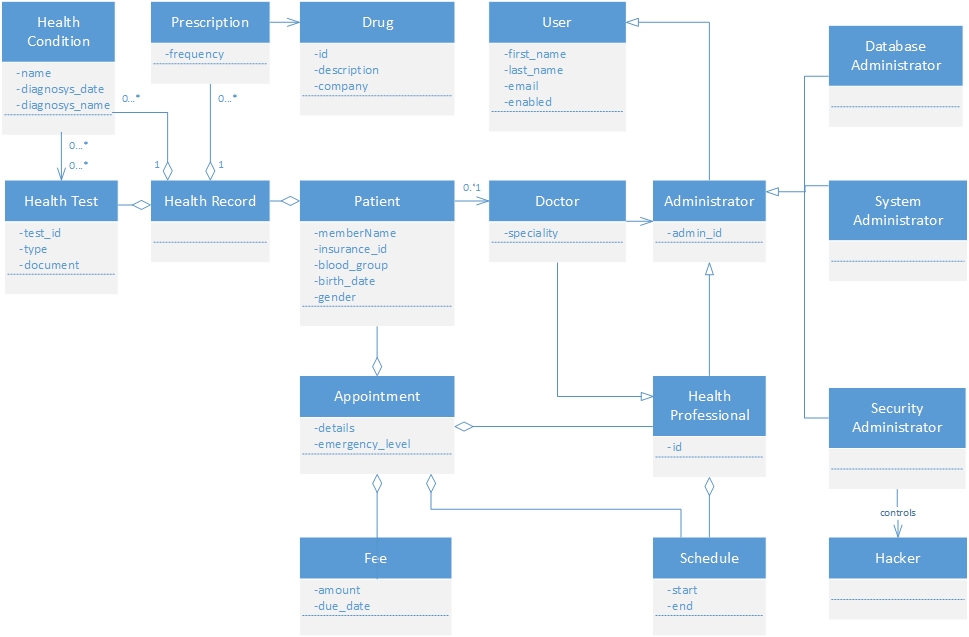


Fig.2. Class Diagram

# **ACTIVITY DIAGRAM**

Activity diagram is a graphical representation of flow of stepwise activities with support for choice, iteration and concurrency. It shows the overall flow of control. The focus of activity modeling is the sequence and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors. These are commonly called control flow and object flow models. The behaviors coordinated by these models can be initiated while previous behaviors have been executed. It happens when objects and data are available or after events occur. Activity diagram is constructed with a limited number of shapes, rounded rectangle representing actions, diamond for decisions, bars for concurrent activities, black circle for start and encircled black circle for end. Arrow runs from start towards the end representing the order of activities.

Flow can be:

* Sequential
* Branched
* Concurrent

iCare activity diagram illustrates the activities involved in using and providing the services of online hospital application. It starts with login for existing user or register for new user. The flow splits into create patient profile for new user and view patient profile for existing user. Branched activities merge into the key activity send patient present health status to doctor. As branched activities can help in choice, doctor takes the help of patient’s old records to prescribe medicine.

Complexity of the application remains low with limited number of activities in the system between patient and doctor. And it does not show any message flow from one activity to another.

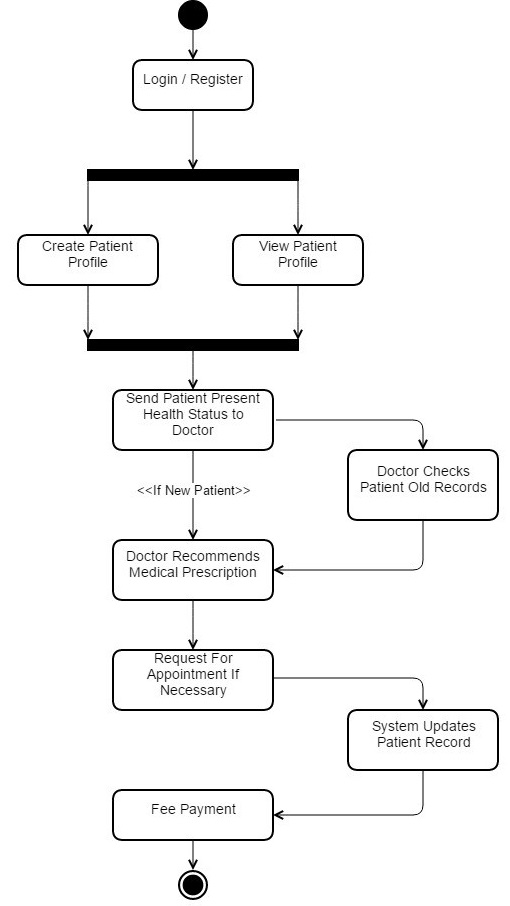


Fig.3. Activity Diagram

# **SEQUENCE DIAGRAM**

Sequence diagram is an interaction diagram that shows how objects interact with one another and in what order. It shows object interactions arranged in time sequence. It also illustrates sequence of messages exchanged between objects. Sequence diagrams are sometimes called event diagrams or event scenarios.

A Sequence diagram shows:

* Parallel Vertical Lines(life lines)
* Processes or objects
* Horizontal Arrows
* Messages

iCare sequence diagram illustrates interaction between three objects. They are Patient, System and Doctor. Initially a patient if new have to register in the system otherwise directly login the system. So, the existing patient will be able to view the previous health record which is stored in the system. Now the patient can send the present health status to the system. So that the system delivers it to the doctor. Doctors can send medical prescription related to the health status of patient. If patient has severe sickness he/she can request for an appointment. After the appointment is scheduled, patient can proceed for payment. The cross lines at the end of life line indicate that the life time of object ended [5].

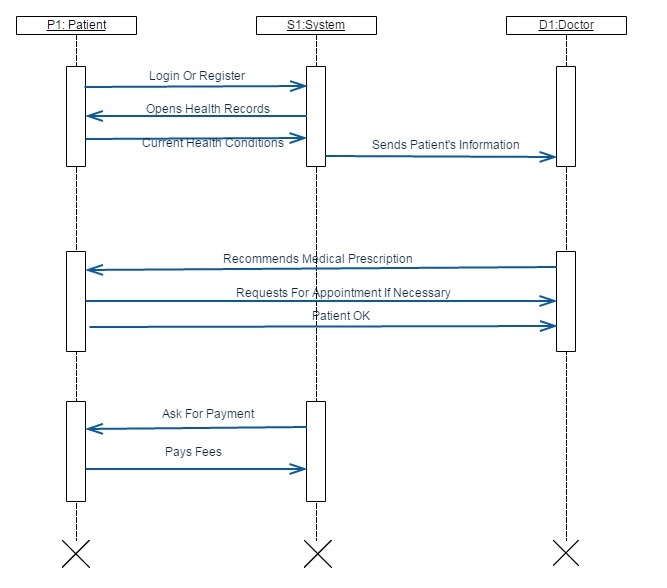
****

Fig.4. Sequence Diagram

# **PACKAGE DIAGRAM**

Packages are UML constructs that enable you to organize, model elements into groups, making your UML diagrams simpler and easier to understand. Packages are depicted as file folders and can be used on any of the UML diagrams, although they are most common on use-case diagrams and class diagrams because these models have a tendency to grow. A package diagram is UML [structure diagram](http://www.uml-diagrams.org/uml-25-diagrams.html#structure-diagram) which shows [packages](http://www.uml-diagrams.org/package-diagrams.html#package) and dependencies between the packages. It is a [namespace](http://www.uml-diagrams.org/namespace.html) used to group together elements that are semantically related and might change together and a general purpose mechanism to organize elements into groups to provide better structure for system model.

Owned members of a package should all be [package-able elements](http://www.uml-diagrams.org/package-diagrams.html#packageable-element). If a package is removed from a model, so are all the elements owned by the package. Package by itself is package-able element, so any package could also be a member of other packages. In addition to the standard [UML dependency](https://en.wikipedia.org/wiki/Dependency_(UML)) relationship, there are two special types of dependencies defined between packages:

* package import
* package merge

A *package import* is "a relationship between an importing namespace and a package, indicating that the importing namespace adds the names of the members of the package to its own namespace."

A *package merge* is "a directed relationship between two packages that indicates that the contents of the two packages are to be combined. It is very similar to Generalization in the sense that the source element conceptually adds the characteristics of the target element to its own characteristics resulting in an element that combines the characteristics of both”.

In iCare package diagram there are multiple packages used such as User, Login, Update Database, Database Administrator and so on that group together elements that are semantically related to the iCare project. [Package import](http://www.uml-diagrams.org/package-diagrams.html#package-import) in iCare system is shown using a dashed arrow with an open arrowhead from the importing namespace to the imported package. By default, the value of visibility is **public**, so it is the same as **«import»**.

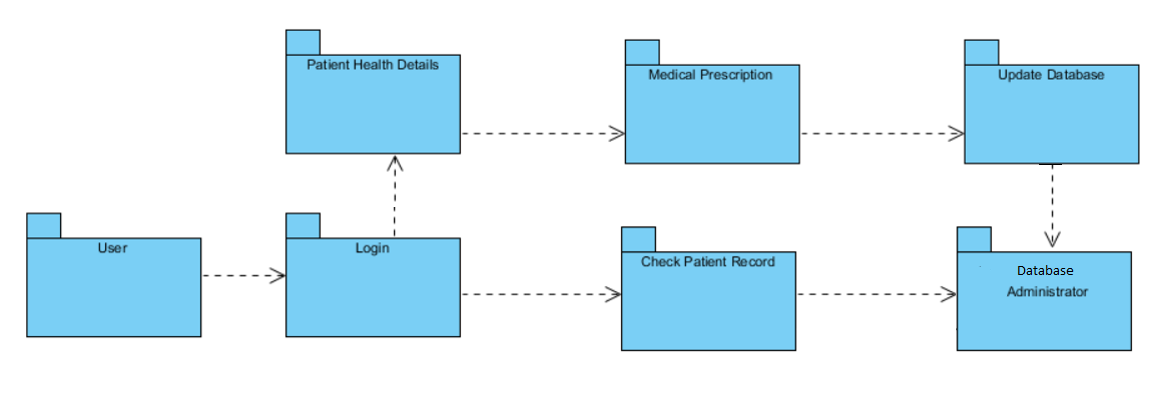


Fig.5. Package Diagram

# **COMPONENT& CONNECTOR VIEW**

C&C diagram illustrates the component and connector structure design of the software architecture. Component is a runtime entity or structure for system functionality such as services, peers, clients, servers, filters, or many other types of runtime elements, connectors are the communication vehicles among components, such as call return, process synchronization operators, pipes, or others. These structures embody decisions as to how the system is to be structured as a set of elements that have runtime behavior (components) and interactions (connectors) and are important for asking questions about system’s runtime properties such as performance, security, availability, and more. This view is excellent for explaining how the system is expected to work and accomplish its job [6].

The C&C design of the system is divided to two main component categories: first category is responsible for the main system’s functionality which includes keeping patients profile, billing, appointment management and reporting. The second category are the components which handles the security aspect of the system.

Since iCare keeps the healthcare records of the patients, information privacy is the first priority of the C&C design. So in this diagram the most important part of the consideration is *security*.

Security is one of the system’s runtime properties that manages and protects the inter-element communication and controls which elements are allowed to access which information; it is a feature of the system that protects data from unauthorized access while still providing access to people and systems that are authorized.

Security has three main characteristics, called CIA: Confidentiality, Integrity and Availability; Confidentiality is the property that data or services are protected from unauthorized access. Integrity is the property that data or services are not subject to unauthorized manipulation. Availability is the property that the system will be available for legitimate use. [6]

The User Account component of the C&C view is responsible for authorization. Authorization grants a user the privileges to perform a task [6]. This component interacts with Security component to provide the users necessary access to run their transactions.

The System Administrator component of the C&C view manages different administration aspects of the system including Security and Database administration so it communicates with Security component.

Other components of this view are the main system functionality structures including Report Manager, User Profile, Billing and Appointment Manager.

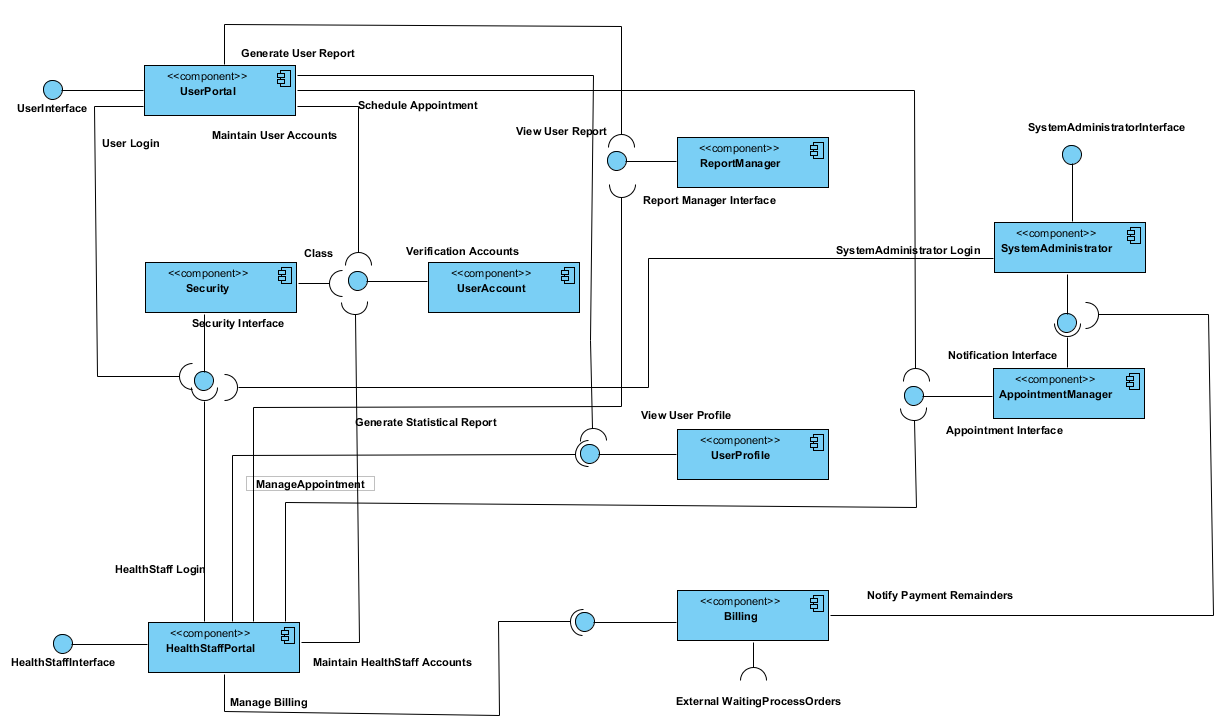


Fig.6. C&C Diagram

# **PATTERNS USED IN THIS ARCHITECTURE**

Patterns are descriptions of a set of predefined subsystems and their responsibilities. It also describes rules and guidelines for organizing the relationships among the subsystems. It is written by people with lots of experience. Patterns make knowledge which could have remained hidden in the heads of these experienced people explicit. This enables others to learn from those experiences. They capture existing, well proven solutions in software development, and help to promote good design practices [7].

Patterns address a recurring design problem, for which a general solution is known among experienced practitioners: a pattern documents existing, well-proved design solutions.

Patterns provide a common vocabulary and understanding of design solutions. Pattern names become part of a widespread design language. They remove the need to explain a solution to a particular problem with a lengthy description. Patterns are therefore a means for documenting software architectures. Patterns support the construction of software with defined properties. When we design a client-server application, for instance, the server should not be built in such a way that it initiates communication with its clients. Many patterns explicitly address non-functional requirements for software systems. For example, the MVC (Model-View-Controller) pattern supports changeability of user interfaces. Patterns may thus be seen as building blocks for a more complicated design [6].

**Differences between design patterns and architectural patterns?**

Design patterns offer a common solution for a common problem in the form of classes working together. They are thus smaller in scale than architectural patterns, where the components are subsystems rather than classes. Design patterns do not influence the fundamental structure of a software system. They only affect a single subsystem. Design patterns may help to implement an architectural pattern [7].

Considering the problems that may occur lead us toward using some patterns in order to overcome those problems. Here the problems and solutions to those problems are categorized and mentioned:

**Problem 1:**

Improving scalability and availability by centralizing the control of resources and services, while distributing the resources themselves across multiple physical servers

**Solution:** Client-server architectural pattern

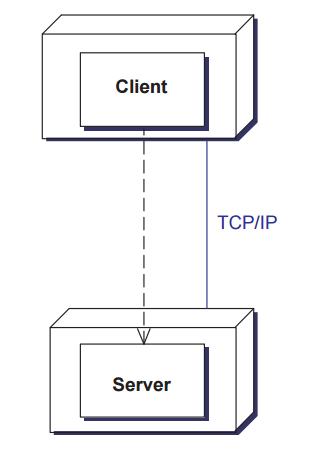
****

Fig.7. Client Server Pattern

In the Client-server architectural pattern, a server component provides services to multiple client components. A client component requests services from the server component. Servers are permanently active, listening for clients. The requests are sent beyond process and machine boundaries. This means that some inter-process communication mechanism must be used: clients and servers may reside on different machines, and thus in different processes. In fact, you can see the Client server pattern as a form of the layered pattern, crossing process or machine boundaries: clients form the higher level and the server forms the lower level [7].

**Problem 2:**

Separating interface functionality from application functionality and yet still be responsive to user input.

**Solution:** MVC (Model-View-Controller) pattern

In the Model-View-Controller pattern, or MVC pattern, an interactive application is divided into three parts: The model contains the core functionality and data, the view displays the information to the user, and the controller handles the input from the user.

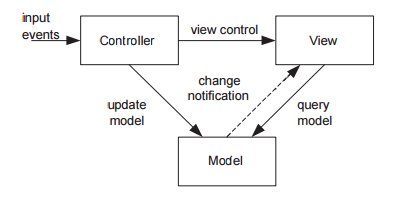


Fig.8. MVC Pattern

The MVC pattern is particularly suitable for multiple graphical user interfaces (GUIs). The model does not depend on the number and kind of GUIs, so the pattern allows for easy changes to the ‘look and feel’. Consistency between model and view is maintained through notification. The MVC pattern often uses the observer design pattern. User input can invoke a change in the model, and a subsequent change in what the view displays [7].

The MVC pattern makes it easy to have multiple views of the same model, which can be connected and disconnected at run-time. It is possible to base an application framework on this pattern. In practice, they are often put together. Views and controllers are also closely coupled to the model [7].

**Problem 3:** The software needs to be segmented in such a way that the modules can be developed and evolved separately with little interaction among the parts, supporting portability, modifiability, and reuse.

**Solution: Layer pattern**

Layered architecture focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each other. Functionality within each layer is related by a common role or responsibility. Communication between layers is explicit and loosely coupled. Layering your application appropriately helps to support a strong separation of concerns that, in turn, supports flexibility and maintainability.

Common principles for designs that use the layered architectural style include:

• **Abstraction:** Layered architecture abstracts the view of the system as whole while providing enough detail to understand the roles and responsibilities of individual layers and the relationship between them.

• **Encapsulation:** No assumptions need to be made about data types, methods and properties, or implementation during design, as these features are not exposed at layer boundaries.

• **Clearly defined functional layers:** The separation between functionality in each layer is clear. Upper layers such as the presentation layer send commands to lower layers, such as the business and data layers.

• **High cohesion:** Well-defined responsibility boundaries for each layer, and ensuring that each layer contains functionality directly related to the tasks of that layer, will help to maximize cohesion within the layer.

• **Reusable:** Lower layers have no dependencies on higher layers, potentially allowing them to be reusable in other scenarios.

• **Loose coupling**: Communication between layers is based on abstraction and events to provide loose coupling between layers [7].

**Context:** All complex systems experience the need to develop and evolve portions of the system independently. For this reason the developers of the system need a clear and well-documented separation of concerns, so that modules of the system may be independently developed and maintained [8].

To achieve this separation of concerns, the layered pattern divides the software into units called layers. Each layer is a grouping of modules that offers a cohesive set of services. There are constraints on the allowed-to-use relationship among the layers: the relations must be unidirectional. Layers completely partition a set of software, and each partition is exposed through a public interface. The layers are created to interact according to a strict ordering relation [8].

iCare application is divided into three separate layers:

* **Presentation layer**: This is the layer through which the users interact with the system. It includes the user interface for the application such as a web application.
* **Business layer**: This layer contains the business process and business components. Business components are based on the business logic of the application which we get benefited from.
* **Data layer**: This layer manages the access to data sources such as health records and interact with data security system and data storage system.

The first box contains the users of our systems such as doctors, patients and those who administrate the system. Moreover, government was considered as an external system which interacts with business layer directly as illustrated in figure 9.

This pattern supports different quality attributes such as more flexibility in adding or removing functionality, increasing reusability due to separation of application into layers, the components in different layers can be reused in similar systems and besides, supporting scalability in accepting more clients. In addition, it provides more facilities in the maintenance of our system by distribution of responsibility and modules in different layers, high cohesion by grouping same functions in one layer and low Coupling by making less interdependency between modules. Security aspects are considerable as well.

Adds-up more complexity and cost to the system are disadvantages of this pattern .Also, layers contributing in penalty performance when a call is made to a function in the top-most layer, this may have to traverse many lower layers before being executed by the hardware. Finally, if the layering is not designed correctly, it may actually get in the way, by not providing the lower-level abstractions that programmers at the higher levels need. [8]

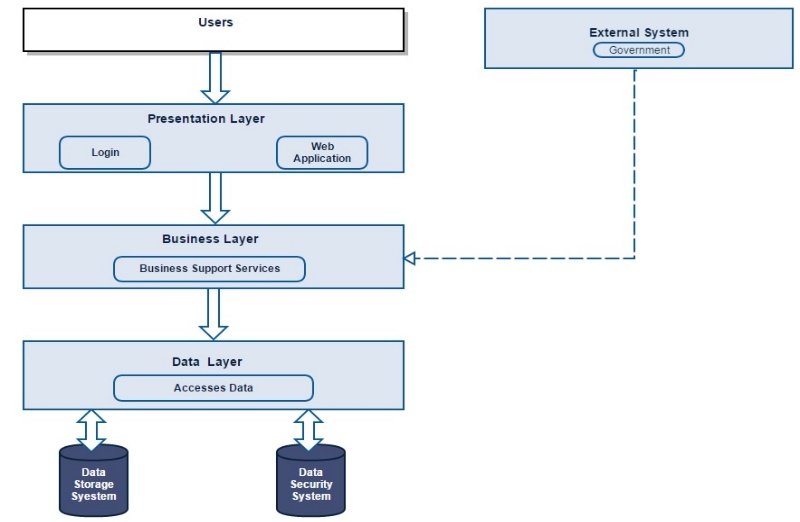


Fig.9. Layer Pattern

# **HIGH LEVEL ARCHITECTURE**

The high level architecture of iCare illustrates all the features built in the iCare software as the components and the Users who are able to view and use the features as modules with ‘uses’ relationship with components shown in the figure 10.

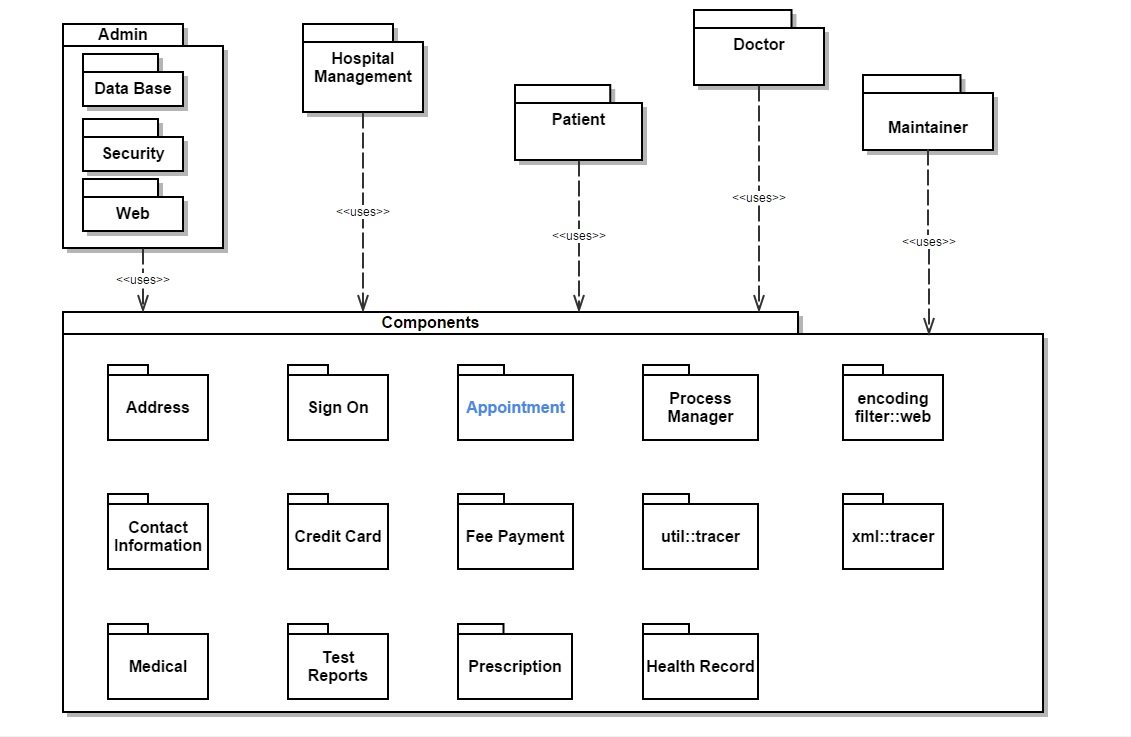


Fig.10. High Level Architecture

**Modification:**

The main modification planned for the iCare new architecture is **“Improvising the availability quality of scheduling appointment”.** The main advantage achieved in the improved feature is the user will be able to cancel his/her appointment at any time and will be able to schedule new appointment without facing any problem. The modification feature is highlighted in the figure 10.

# **CONCLUSION**

iCare is designed in such a way that it can provide all online hospital services easily. The automation of the entire system improves the efficiency of the project. Any information system is a complex entity formed from various parts with one common plan and purpose. A critical feature is the interdependency, and interactions between, those who collect data and process it into information and evidence, and those who use the information. iCare provides a friendly graphical user interface which proves to be better and it overcomes the delay in communication. A successful health information system should connect and respond to their needs. A formal and complete health information system infrastructure with accurate reporting of all aspects of health in a verifiable way is best, but costly and time consuming to create, and in many instances impossible to implement. High-quality health information is a critical input into clinical, local, national and global decision-making [9]. It also gives appropriate access to the authorized users depending on their permissions. System security, data security and reliability are the striking features. iCare has adequate scope for the modification in future if necessary.

**GLOSSARY**

|  |  |
| --- | --- |
| **SAD** | Software Architecture Description |
| **HIS** | Health Information System |
| **EHR** | Electronic Health Record |
| **UML** | Unified Modelling Language |

**References**

[1] <http://www.who.int/healthinfo/statistics/> toolkit\_hss/ EN\_PDF\_Toolkit\_HSS\_ServiceDelivery.pdf

[2] <http://www.cpc.unc.edu/measure/our-ork/health-information-systems>

[3] D2.2 Privacy & Security Legal Issues Report (2014). Ross Little. Daphn

[4][http://www.iso-architecture.org/ieee- 1471/templates/42010-ad-template.pdf](http://www.iso-architecture.org/ieee-%20%201471/templates/42010-ad-template.pdf)

[5] <https://en.wikipedia.org/wiki/Sequence_diagram>

[6] Software Architecture in Practice, 3rd edition, Len Bass, Paul Clements, and Rick Kazman, Addison-Wesley, 2013, ISBN 9780321815736. (Required)

[7] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, 3rd ed. Upper Saddle River, N.J.: Addison-Wesley, 2013.

[8]http://portal.ou.nl/documents/114964/2986739/IM0203\_03.pdf

[9] <https://msdn.microsoft.com/en-ca/library/ee658117.aspx#LayeredStyle>

[9] <http://www.uq.edu.au/hishub/docs/WP05/HISHUB-WP5-Sum-07-WEB%2015-04-13.pdf>ion improvement, computer-assisted coding, mobile physici