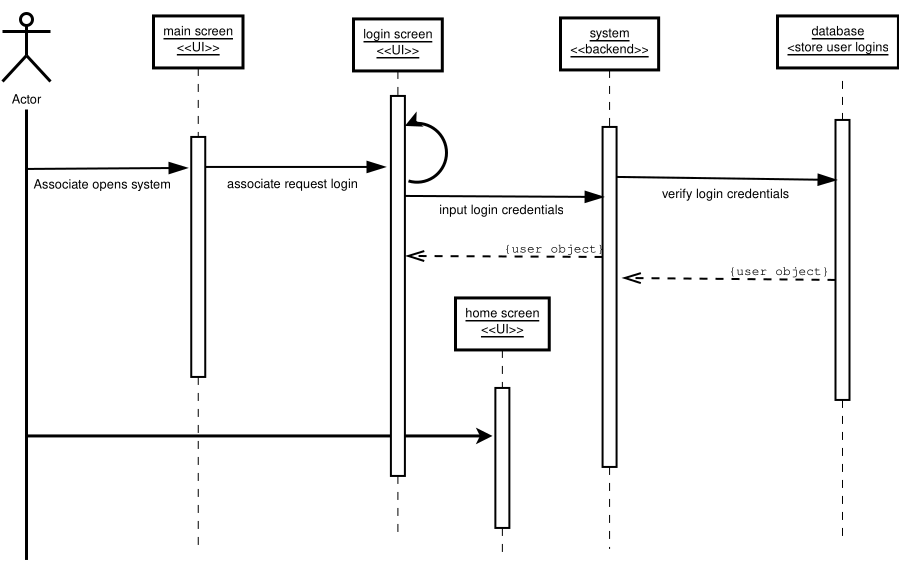
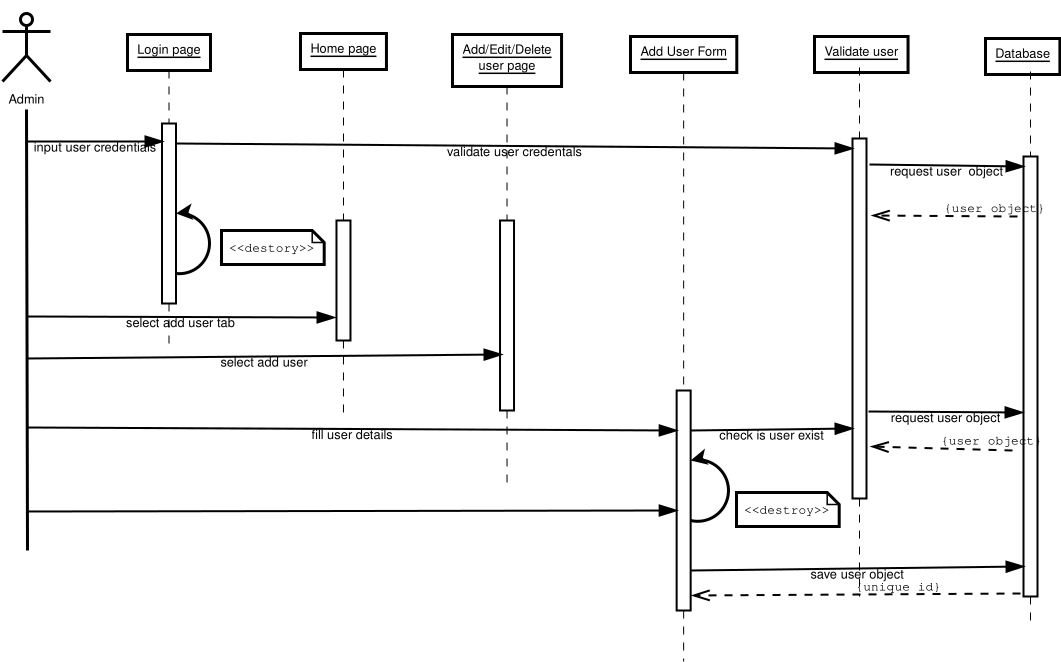
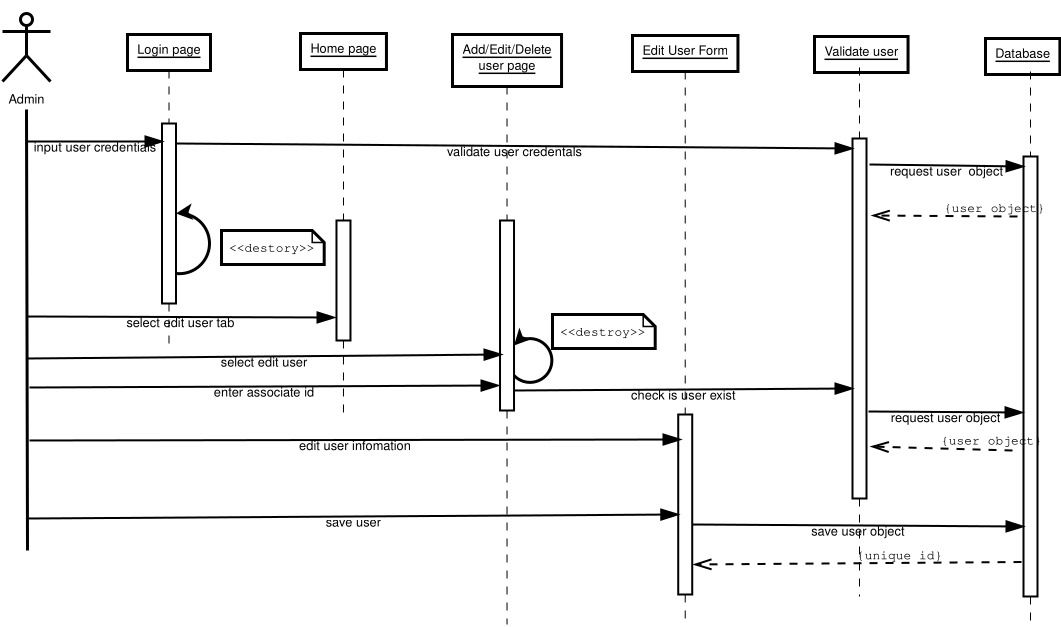
**SYSTEM DESIGN AND ARCHITECTURE**

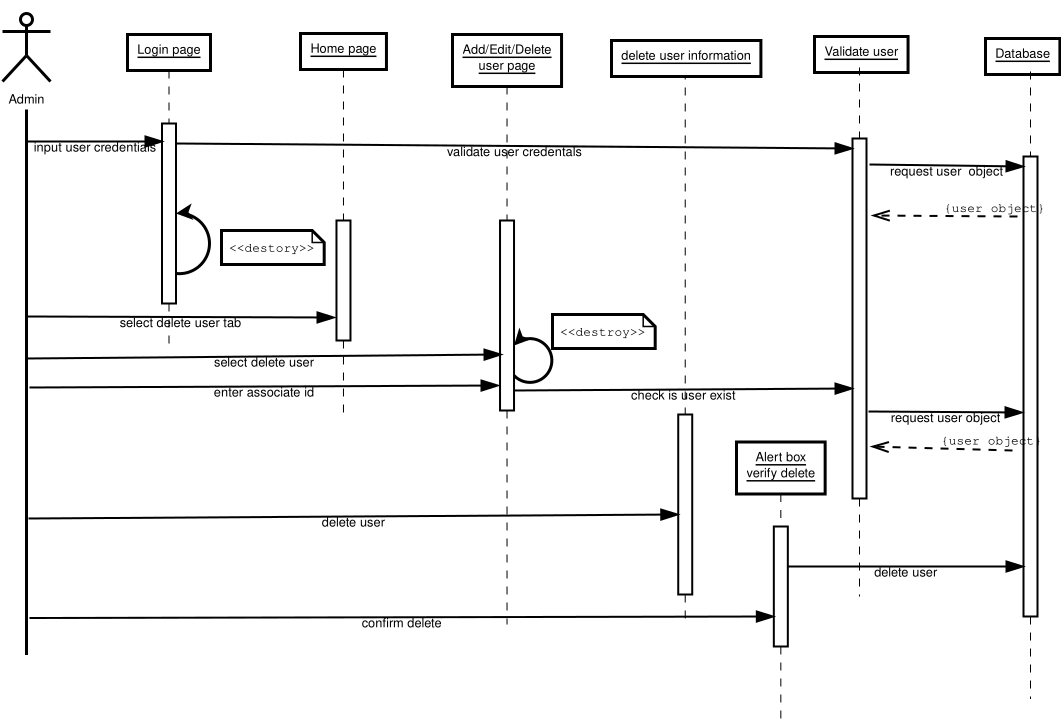
1. Interaction diagrams.

UML ( unified modeling language ) is the defacto way of representing software systems visually. To visualize the system 'fully dressed' use case scenarios 4 sequence diagrams were designed, the diagrams below represent each use case.

Figure 1.1 login sequence diagram.

Figure 1.2 represents the add user use case.

Figure 1.3 represents the edit user sequence diagram

Figure 1.4 represents the delete sequence diagram.

The above sequence diagrams paint a sequence of event scenarios that describes how the main actors interact with the system. In figure 1.1 it shows the event process of login into the system the associates are required to provide their login credentials namely user name and password. Figure 1.2 depicts the event scenario that the administrators follow to add a user to the database the input required for this event is the associate details, the associate is saved to the database and given a unique identification id.

Figure 1.3 represents the edit user sequence diagram that visualizes the process of editing user-related information the input for this process is the associates' id which allows the administrator to pull the associates' information from the database. The delete sequence is represented by figure 1.4 the administrator need only to input the associate id to retrieve the user object from the database, the administrator is given a chance to confirm the deletion in the form of a modal alert box the administrator can then acknowledge or reject the deletion.

**Design Principles.**

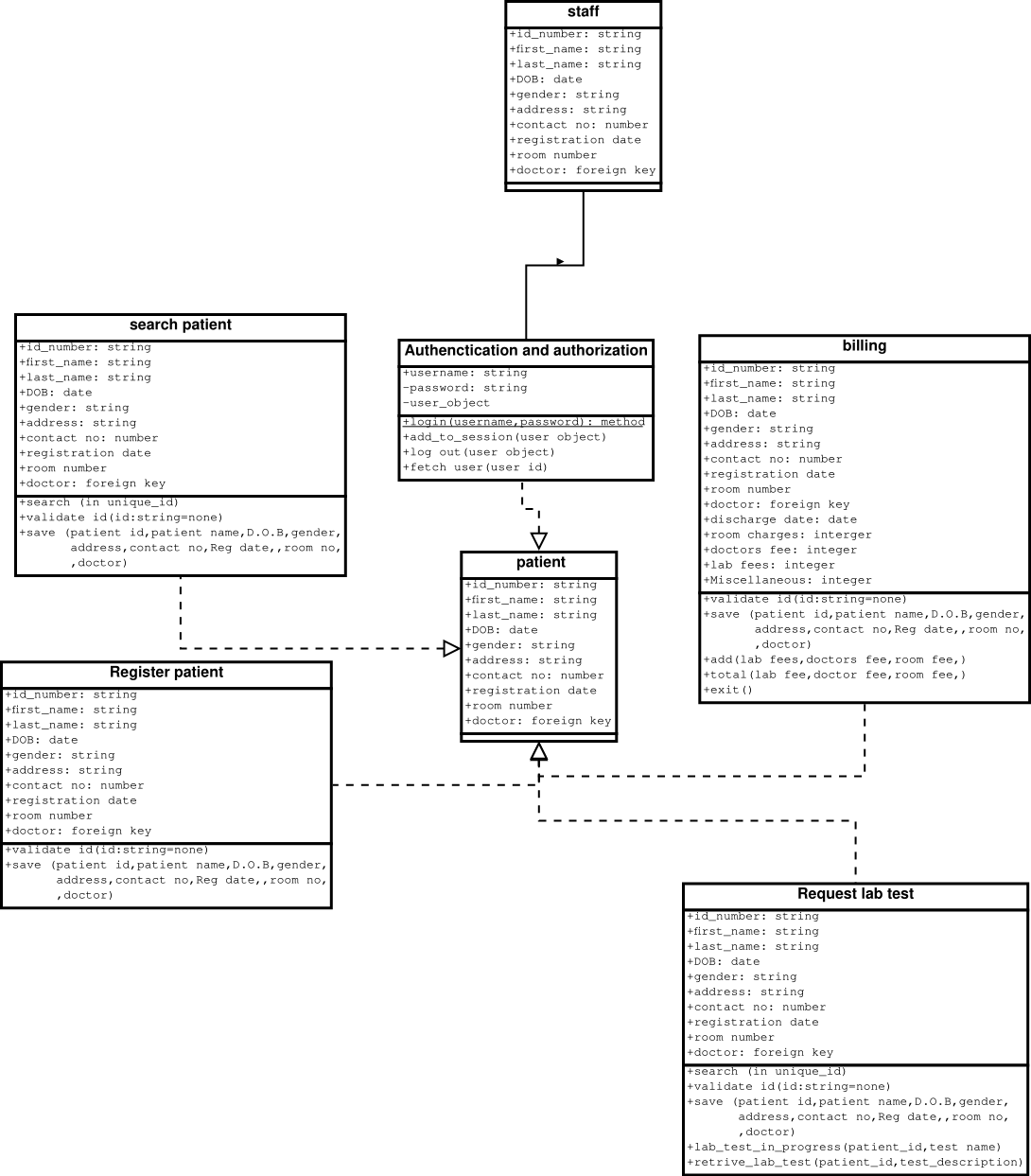
To make the system more flexible, maintainable and understandable, several design principals are incorporated into the system design namely;

1. SOLID concepts I.e the single responsibility principle.
2. Composition.
3. Don't repeat yourself (DRY) principle.

Solid design is a bundle of concepts, in our system, we use the single responsibility principle, which dictates that a class should only handle one responsibility, for example, the authentication class which is responsible for login in and out the user. To mangle all these classes into a system we have used the composition principle over inheritance in building the structure of the system. The composition further allows hot-plugging of a new feature to the system as each composition is a class of its own. Being that the different classes implement different functionalities, this feature greatly works to our advantage as it helps keeps our code DRY by creating abstraction layers.

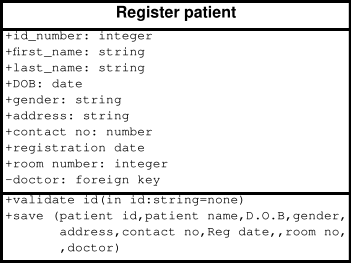
Section 2: Class diagram and interface design.

Class diagram.

Figure 1.5

**Data types and operation signatures.**

The class diagram below visualizes the register patient function in our system. The class solely handles the process of registering a patient into the system, this class uses the save operation to store patient-related data in the database and returns a unique id that will be used to identify the patient in the facility. This event scenario is visualized in the register patient use case.

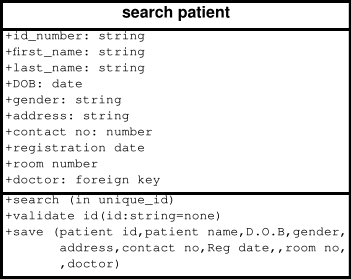


The Register patient class has the following attributes and operations;

|  |  |
| --- | --- |
| **Attribute** | **Data type** |
| first\_name | string |
| last\_name | string |
| Date of birth | date |
| gender | string |
| Contact no | integer |
| Registration date | date |
| Room number | integer |
| doctor | integer |

|  |  |
| --- | --- |
| **Operation** | **Return type** |
| Save method | Integer (Unique id) |
| Validate method | Bool (Check if the user already exists) |

The search class handles the responsibility of searching for a patient in the database or an associate by id , figure 2.1 describes the search class with all its attributes and operations, the class further allows the user to save edit associate or patient information.

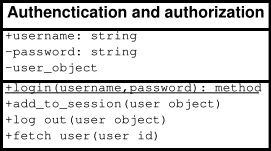
Figure: 2.2

The search class has 10 attributes and three operations listed in the tables below respectively,

|  |  |
| --- | --- |
| **Attribute** | **Data type** |
| first\_name | string |
| last\_name | string |
| Date of birth | date |
| gender | string |
| Contact no | integer |
| Registration date | date |
| Room number | integer |
| doctor | integer |

|  |  |
| --- | --- |
| **Operation** | **Return type** |
| Save method | Integer (Unique id) |
| Save method | Object (user object) |
| Validate method | Bool (Check if the user already exists) |

The authentication and authorization class is in charge of handling all functions related to log-in, log-out and session authentication, this class implements the security module of the system limiting access to different parts of the system take for example the receptionist cannot add or delete users from the system because his/her security clearance does not allow him/her to use that functionality only the administrator can add or delete user, figure 2.3 shows the authentication and authorization class with all its attributes and operations.

  
Figure: 2.3

Authentication and authorization class has two private attributes and one public attribute. The class also has four operations listed in the table below.

|  |  |
| --- | --- |
| **Attribute** | **Data type** |
| Username | String |
| Password | String |
| user\_object | Object |

|  |  |
| --- | --- |
| **Operation** | **Return type** |
| login | User object |
| Add to session | Session id |
| Log out | none |
| Fetch user | User object |

The billing class handles the billing for all patients admitted to the facility, it uses the unique id to look up the patient in the database, Charges range from lab fee, room fee to doctors fee. The class is in charge of summing all these charges up and providing the patient or guardian with the invoice of all charges. Figure 2.4 displays the billing class.

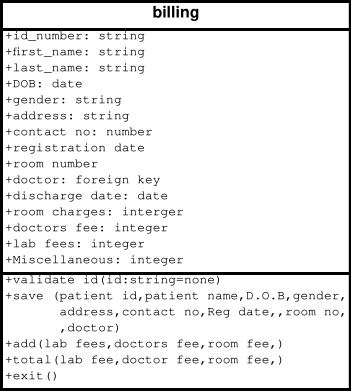


Figure 2.4

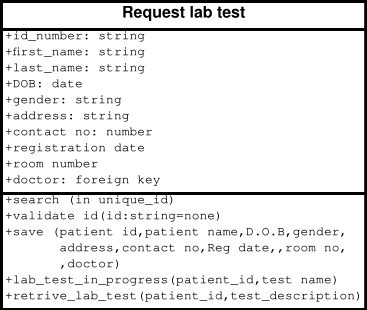
The table below shows all the attributes and operations in the billing class.

|  |  |
| --- | --- |
| **Attribute** | **Data type** |
| Id number | integer |
| first\_name | string |
| last\_name | string |
| Date of birth | date |
| gender | string |
| Contact no | integer |
| Registration date | date |
| Room number | integer |
| doctor | integer |
| Discharge date | date |
| Room charges | integer |
| Doctors fee | integer |
| Lab fee | integer |
| Miscellaneous | integer |

|  |  |
| --- | --- |
| **Operation** | **Return type** |
| Save method | Integer (Unique id) |
| Validate method | Bool (Check if the user already exists) |

The Request lab test handles all operations regarding, requesting of lab test, displaying lab test results, and saving lab test results to the database. This class is used by both doctors and lab technicians, a doctor can place a request for lab test through this class and the lab technician can be able to see the request and act on it accordingly. Figure 2.5 describes the request lab test.

Figure: 2.5

  
Figure 2.5

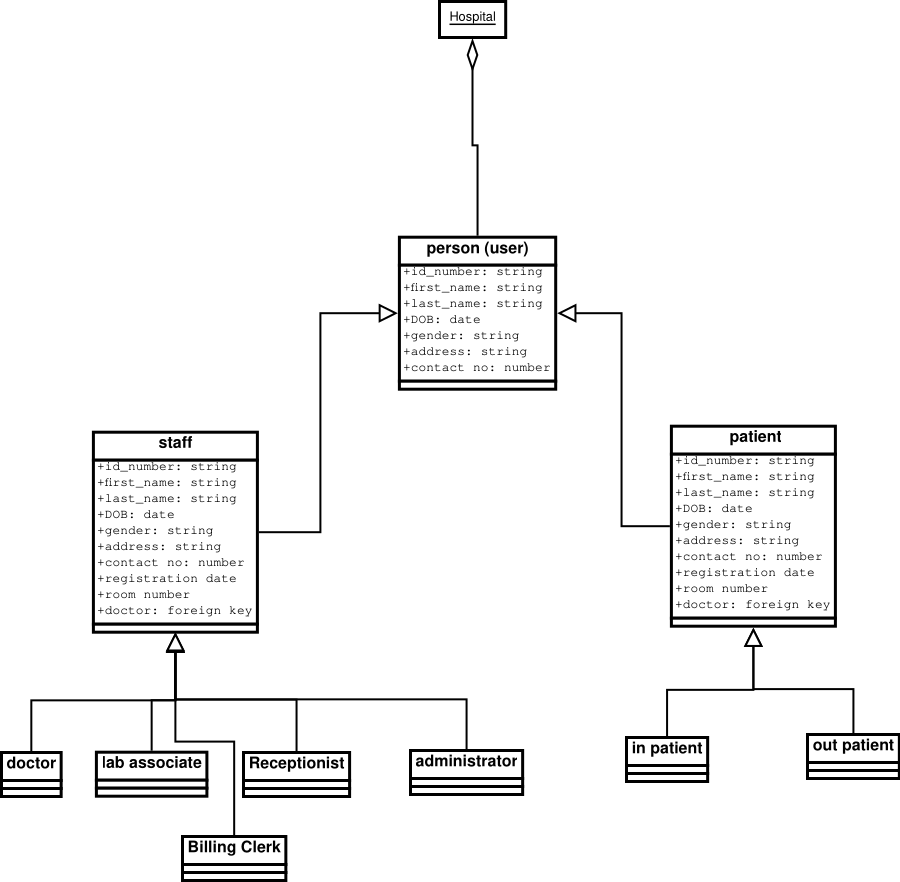
The table below shows all the attributes and operations in request lab test class,

|  |  |
| --- | --- |
| **Attribute** | **Data type** |
| first\_name | string |
| last\_name | string |
| Date of birth | date |
| gender | string |
| Contact no | integer |
| Registration date | date |
| Room number | integer |
| doctor | integer |

|  |  |
| --- | --- |
| **Operation** | **Return type** |
| Save method | Integer (Unique id) |
| Save method | Object (user object) |
| Validate method | Bool (Check if the user already exists) |

**Traceability Matrix**

**Domain Model**

 Figure 3.1 shows the domain model of our system the staff and patient classes stems from this model, the staff class holds all data related to the hospital associates for example doctors and lab associates, to allow associates to search through the database with ease we implement a search class that handles searching the database to retrieve data related to either patients or associates. To facilitate billing within the hospital the billing class was implemented this allows the billing clerk to issue an invoice to patients and keep track of all the bills within the facility. The authentication and authorization class implement the security requirements of the system keeping track of all users in the system, this class also prevent unauthorized access to data related to patients or associates by third parties.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Req’t | PW | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 | UC8 | UC9 | UC10 | UC11 | UC12 |
| REQ1 | 5 | X |  |  |  |  |  |  |  |  |  |  |  |
| REQ2 | 15 |  |  |  |  |  |  |  | X |  |  |  |  |
| REQ3 | 15 |  |  |  |  |  |  |  |  |  | X | X | X |
| REQ4 | 5 |  | X |  |  |  |  |  |  |  |  |  |  |
| REQ5 | 15 |  |  |  |  |  |  |  |  | X |  |  |  |
| REQ6 | 5 |  |  | X |  | X |  |  |  |  |  |  |  |
| REQ7 | 10 |  |  |  | X |  |  | X |  |  |  |  |  |
| REQ8 | 5 |  |  |  | X |  | X |  |  |  |  |  |  |
| REQ9 | 10 |  |  |  | X |  |  | X |  |  |  |  |  |
| **MAX PW** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **TOTAL PW** |  | 5 | 5 | 10 |  | 5 | 5 | 10 | 15 | 15 | 15 | 15 | 15 |

**System Architecture and design**

The system architecture is a conceptual model that describes a system, it defines the behavior, structure, and views of the system. The system architecture can consist of subsystems and system components.

**Architectural style**

The system uses client-server architecture thus allowing the system to be distributed within the facility, take for example the administrator can have his/her client that interfaces with the server. This architectural style allows the server to be in a centralized place and the client can be in different departments within the hospital facility this greatly works to our advantage because it allows the system to grow both horizontally and vertically. In the future in the hospital needs to add a new department all that is need is to connect the client to the server. To future make the system flexible both the layered and component-based approach was used this allows the system functions to be sub-divided adhering to the design principles of the project which state that each class within the system can only handle one responsibility take for example the search class which implements the search functionality in the system the class can the accessed by other classes needing the search functionality

The server side is divided into three tiers each with its responsibility namely;

* the business logic unit, this unit is in charge of all the business logic, for example, calculating the bill
* database, the database is in charge of storing all the hospital-related data
* Authentication and authorization unit this unit handles all the sessions within the system thereby providing security.

**Subsystems**

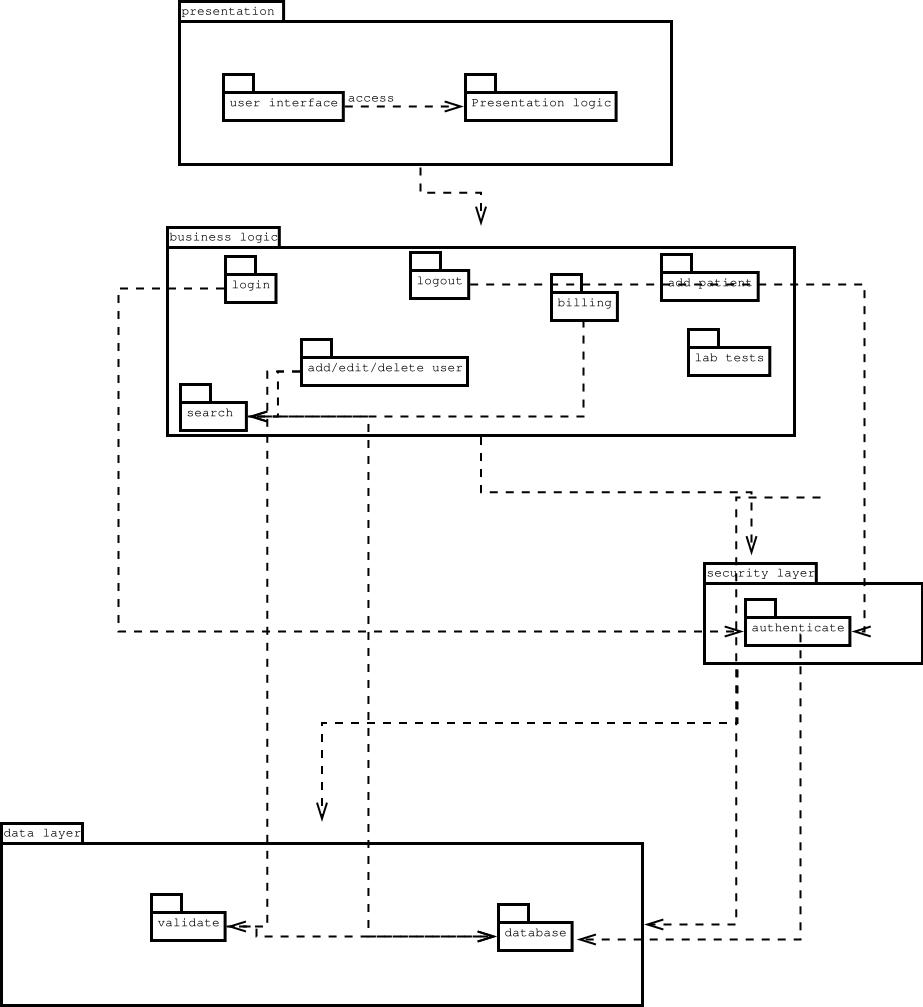


Figure 4.1

**Mapping subsystems into hardware.**

Our system architectural style is majorly based on the server-client model, the presentation subsystem will sit the client-side and the rest of the subsystems will sit on the server-side. On the client-side, the standard user needs to have a personal computer running any browser. On the server-side we need to have two bare metal servers the first one running our data subsystem, while the second one running the authentication and business logic subsystems, this setup allows the system to run faster because of the distribution of tasks, the setup further boots security because of the layering of security.

**Persistent Data Storage.**

The system uses a relational database to store data, the data split into tables following the entity relation diagram namely;

* staff table
* patient table
* billing table
* room table

each of these tables has their attributes that describe the entities in these tables.

**Network Protocol**

The server side uses sockets to connect between the database and the business logic subsystems each running on a different server. The network protocol used is TCP protocol which allows end to end connection between the two. The connection between the client and the server uses the secure https protocol. In https protocol the data is encrypted.

**Global control flow**

The system is event-driven the server is always listening for connections when a client connects the server creates a thread to handle the needs of that client and continues to listen for incoming connections, because of threading the different clients can execute different tasks within the system this removes the time dependency. Also in the data subsystem, the database spins off threads to handle the different requests by the client through the business logic subsystem. To maintain the integrity of the system the threads make use of semaphores, mutexes, and locks. These controls prevent two threads from writing to the same object at the same time hence corrupting the data.

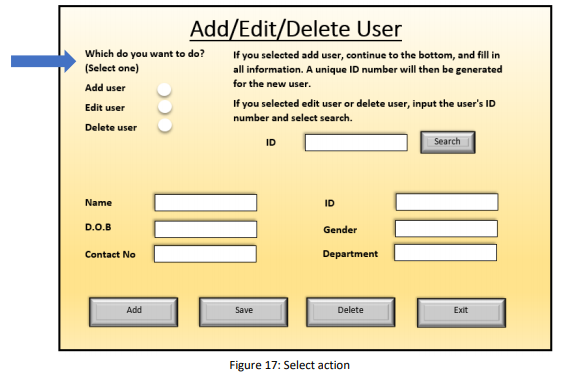
**Hardware requirements**

The client-side does not need any specific hardware to run a standard personal computer should suffice, but on the server-side, we need to use machines with the following requirements;

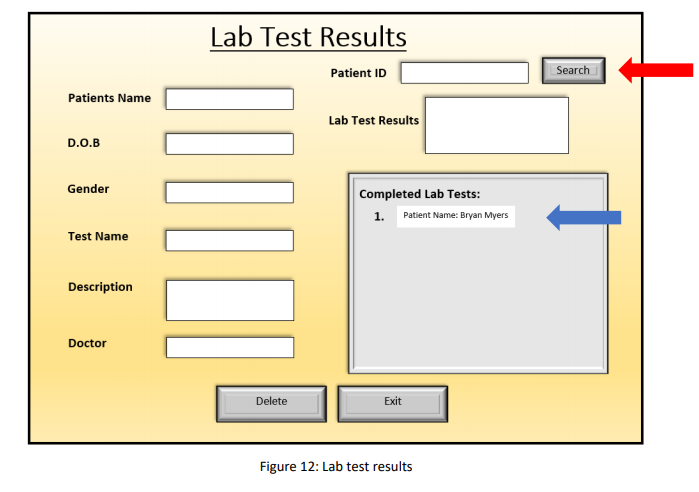
* CPU with at least 6 cores.
* 16 GB of RAM.
* One Gigabyte capable network card.
* 1 Terabyte solid-state drive.
* 4 Terabytes hard disk.

The above requirements will make the system have a faster response time because the CPU can handle all the incoming client request, the RAM makes the machine handle a lot of processes at a time this also allows users to fully maximize the CPU, for the one-gigabyte solid-state drive this make our reads and write to the database even faster. The network card allows the system to use internet connections of over one gigabyte thereby increasing the overall response time of the system.

**User interface design and implementation**



To further improve the user interface the search interface and the edit user information where all bundled up in to a single system this allow ease of navigation within the system all all these functions are closely related.



The lab test user interface was further refined allowing the doctor to post new request, view on going test and completed tests. The user interface was implemented in such a way because all the functionalities are closely related.

**Test design**

Testing of the system is divided into two unit testing and intergration testing, unit testing runs tests on the individual modules and classes while intergration testing test how well the individual components perform as a complete system, to test the system we have used test runner to execute the test cases automatically thus saving us the effort of running the test manually. The tests are encapsulated in test classes each test being a method of the test class this help keep the test simple and easy to understand.Our tests are divided into four classes namely;

* test search class
* test register class
* test edit/add/delete class
* test billing class

**Project Coordination and Progress Report**

i. List use cases that have been implemented.

All usecases from UC1 to UC12

1. What is already functional, what is currently being tackled?

The entire project is already functional. What remains is project handover and deployment

1. List and describe other related project management activities.

Project deployment and maintenance

b. Plan of Work

i. List the project milestones and dates by which you plan to accomplish them using Gantt chart

1. Software and hardware acquisition- 4 months
2. Project design- in 6 months
3. Project testing- 11 months
4. Project handover-in 12 m

References

visual paradigm. (n,d). what is a package diagram . visual paradigm. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/

visual paradigm. (n,d). what is a class diagram . visual paradigm. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/>

wikipedia. (n.d) domain model. Wikipedia <https://owl.purdue.edu/owl/research_and_citation/apa_style/apa_formatting_and_style_guide/general_format.html>

wikipedia. (n.d) SOLID. Wikipedia https://en.wikipedia.org/wiki/SOLID

wikipedia. (n.d) Composition over inheritance . Wikipedia https://en.wikipedia.org/wiki/Composition\_over\_inheritance

Wikipedia . (n,d) don’t repeat yourself. Wikipedia [https://en.wikipedia.org/wiki/Don%27t\_repeat\_yourself](https://en.wikipedia.org/wiki/Don't_repeat_yourself)