Vaccine Management System

(Drug_Lord_v.1.10)

Index

Project Synopsis

Title.

Vaccine Management System Codename: DrugLordv1.10

Problem Statement

The main objective of this System is to maintain records of vaccinations to monitor the quality of vaccines and timely assurance in rural areas where health centres are not easily accessible.

Why this Topic?

To reduce the paperwork required for maintaining records instead digitizing them for data visualization to minimize reading of records and provide easy access to third-party bodies to get, deliver, and administer vaccines.

Objective and Scope.

- To reduce the paperwork required
- To maintain a detailed record of each individual with ease
- To analyse data easily for the person/body using this system
- To create a system that manages both deliveries along with a facility for vaccine administration

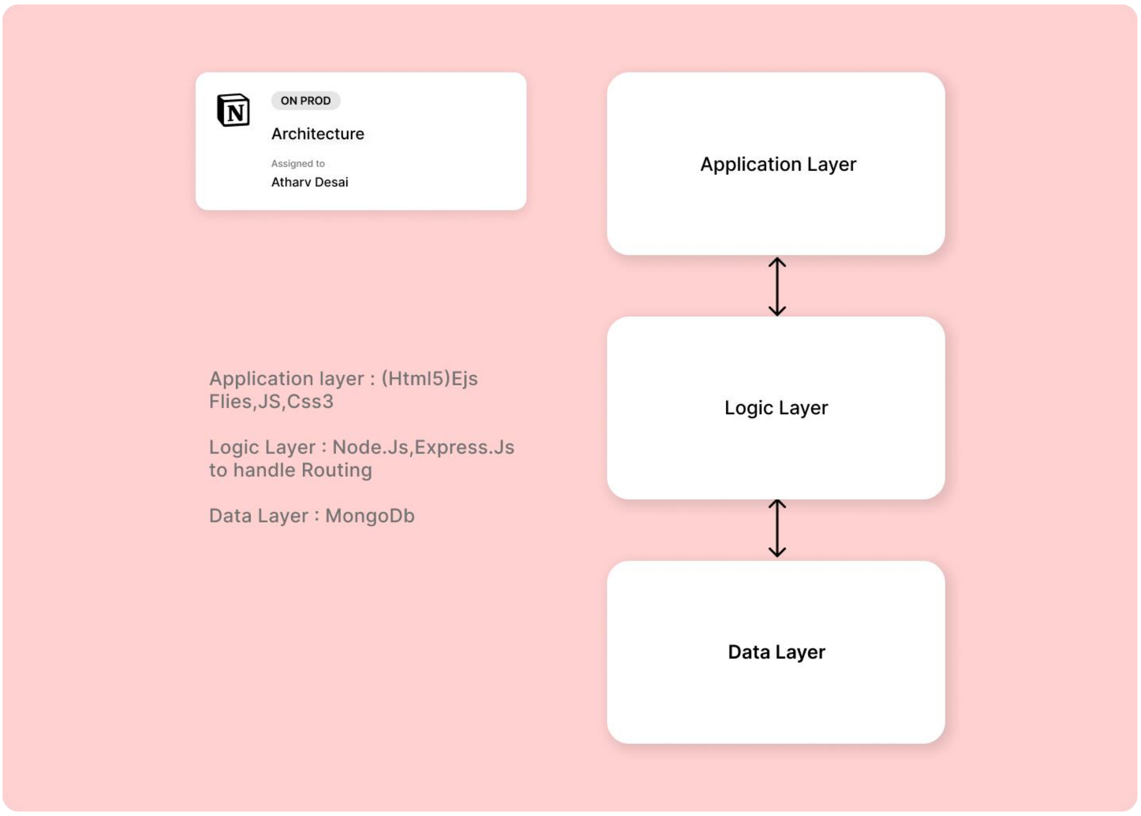
Figure 1.7.1

Methodology for developing project.

In this system, I am going to use Extreme Programming for developing an appropriate system as a solution for rapidly changing requirements

Advantages: Communication, Simple, Easy, Agile.

Proposed Architecture



Requirements

Software Requirements

- Front-end : HTML5, CSS , JS,Tailwindcss
- Back-end :,ExpressJs, NodeJS, MongoDb.
- Operating System: Windows 7.0 +.

Hardware Requirements

- Processor: Intel Core Duo 2.0 GHz or more.
- RAM: 2 GB or more.
- Monitor: 17 CRT or LCD.
- Hard disk: 500 GB or more.
- Keyboard: Normal or multimedia.

Platform

Visual Studio Code

Contribution

This system will help people in rural areas get access to vaccines along with reduced paperwork and record generation and keep each individual to date.

Conclusion

This system will help to reduce paperwork and provide access to people in rural areas.

1. Introduction

1.1 Background

The current Vaccination Systems which are used are not centralized i.e., the data generated is confined to an organization or a company except due to recent pandemic newer methods are cultivated

References

- https://www.servicenow.com/solutions/vaccine-management.htm
- https://www2.deloitte.com/us/en/pages/public-sector/solutions/vaccine-management-system.htm
- https://www.intelex.com/products/applications/vaccine-management}{Intelex Vaccine Solution

1.2 Objective

The main purpose of this project is to help benefit NGOs in providing vaccine access to rural areas people without using many resources, by providing verified personnel of NGOs or any individual to request access to medical vaccines under the guidance of a medical practitioner.

1.3 Purpose

- Provide rural areas with exposure to vaccine
- Create a platform where people can register for an appointment, and delivery as well as monitor the records.

1.4 Application

The idea can be fundamentally used in any management system of products in any medical field.

1.5 Scope

Creating a platform for all domestic producers to sell their vaccines to local consumers, NGOs, etc, and to simplify the delivery process, regulations and management.

1.6 Achievements

Through this project, I will get to learn various technologies used to develop a system. This system will give a digital platform to the all-Domestic producers of vaccines which will increase the vaccine production rate and simplify the delivery process as well as help them to manage their data. Once they get satisfactory results from the system, users will start trusting the system and eventually, most of the tasks will get digital. Encourage B2B and C2C relations

2. Survey of Technology

The number of Technologies available for the implementation is listed below

- ASP.NET(MVC) Core 5.0+
    - Is a framework developed by Microsoft to create enterprise-grade web apps in a short amount of time
    - Languages : C#, HTML, CSS, jQuery
    - Backend: Microsoft SQL Serve
- Django (MVC)
    - Framework to create scalable single-page web apps,web-API using Python as the main programming language
    - Language : Python,HTML,CSS,jQuery,JS
    - Backend: MongoDB
- Ruby on Rails
    - Framework based on Rudby Programming Language for easy file directory structure and dataflow subroutines
    - Language: Ruby,HTML,CSS,JS
    - Backend: MySql Server
- Node.js
    - Backend as well and Frontend using vanilla Javascript as main primary language useful in API creation
    - Language: JS (EMAC5), HTML(ejs), JS,jQuery
    - Backend: MongoDB
- Flask
    - Python Backend Framework, useful in developing fast prototypes of ML and AI searches, etc
    - Language: Python

For my current project, I am going to use Node.js as a development platform for easy implementation of the requirements proposed.

Why Node.js?

Easy routing through Express.JS library and ejs content to render with minimum requirements to hosting a server and is platform independent

Required: Node.js is installed

3. Requirement and Analysis

3.1 Problem Definition

What to expect about the system?

The system will manage and record all the deliveries along with orderly administration of vaccines It will also keep a detailed record of all the domestic suppliers, users requesting appointments, providers buying vaccines, and local company producers producing vaccines

- [Users, Providers, Producers]
    - Will be allowed to log in as each unique individual account.
- [Producers, Providers]
    - Help manage, and administer vaccine drives with all the information centralized for easy fetch
    - The system will keep a detailed record of all individual's information both registered for vaccine drives as well as seeking appointments
- [Provider]
    - To set up appointments for the user or use for delivery purposes.
- [Producer]
    - Set up vaccine stocks for order processing
- [User]
    - Book a nearby appointment for quick administration

3.1.1 Sub Problems

- For Users
    - How users can add their information?
    - How to schedule and select an appointment?
    - How to get the vaccine?

- For Providers
    - How to get verified?
    - How to select your order?
    - How to schedule appointments for the people?

- For Producers
    - How to set up your vaccine Stocks?
    - How to process Orders?
    - How to Deliver?

3.1.2 Problem Description.

The system will help find required clients for domestic producers as well as enable recipients to get vaccine appointments in their nearby locality

3.2 Requirements Specification

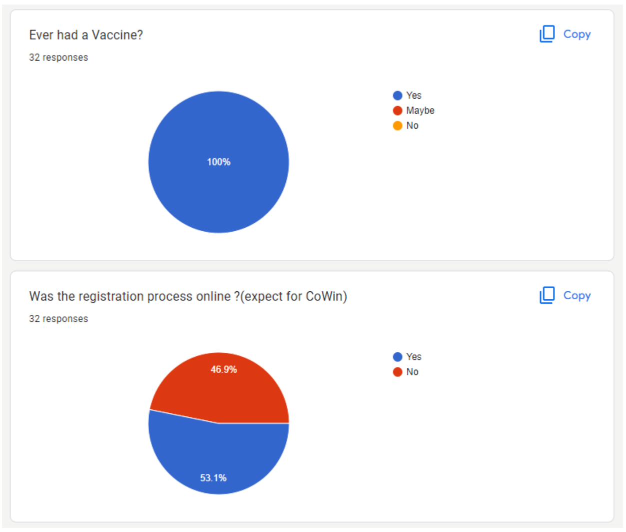
3.2.1 Requirement analysis

Identify stakeholders i.e., in case the people who are going to use this app

- Capture requirements
- Holding One-One interviews
- Conduct Group Workshops
- Get Feedbacks
- Build Small Prototypes

For the current situation, I used the Feedback method to identify the requirements for the project using Google Forms as a means to collect the data

The below figures are the collected data that was generated.

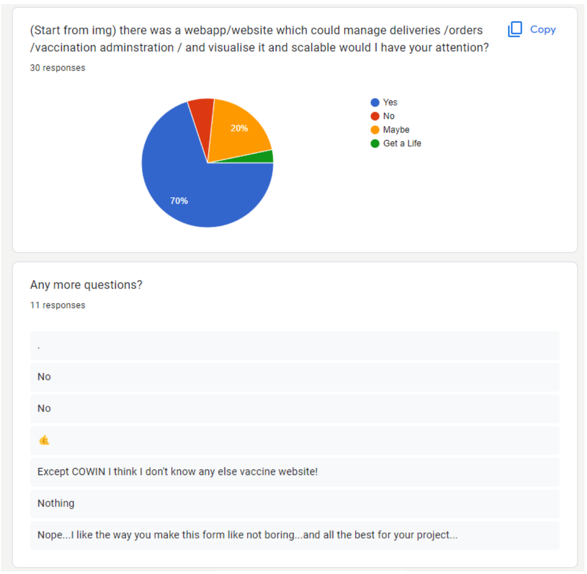

Ever had a Vaccine?

32 responses

Was the registration process online ?(expect for CoWin)

32 responses

Figure 3.2.1


(Start from img) there was a webapp/website which could manage deliveries /orders /vaccination adminstration / and visualise it and scalable would I have your attention?

30 responses

Any more questions?

11 responses

.

No

No

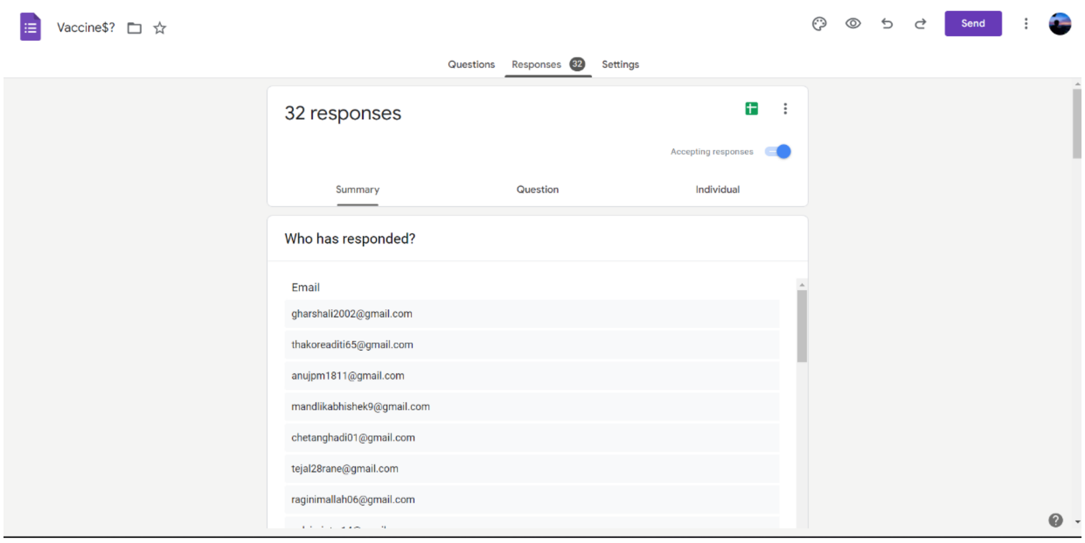
👍

Except COWIN I think I don't know any else vaccine website!

Nothing

Nope...I like the way you make this form like not boring...and all the best for your project...

Figure 3.2.2



32 responses

Who has responded?

Email
gharshali2002@gmail.com
thakoreaditi65@gmail.com
anujpm1811@gmail.com
mandlikabhishek9@gmail.com
chetanghadi01@gmail.com
tejal28rane@gmail.com
raginimallah06@gmail.com

Figure 3.2.3

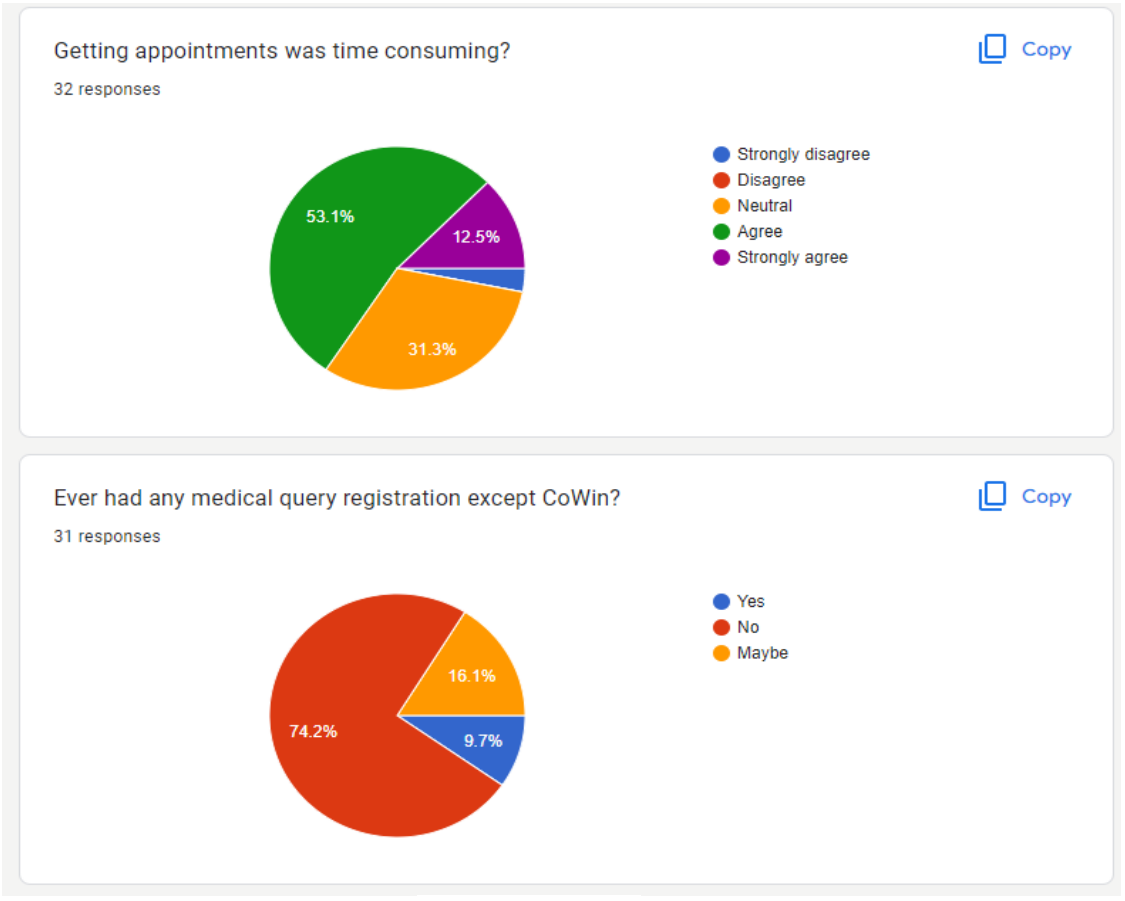


Figure 3.2.4

3.2.2 Functional Requirements

1. The system should allow users to fill up for an appointment in their locality
2. The system should keep a detailed record of user's info
3. The system should allow users to note any problems after vaccination is completed
4. The system should provide Producers, NGOs, and Companies to order and sell their vaccines

5. The system should keep a detailed record of all domestic producers, providers and along with the info of users

3.2.3 Non-Functional Requirements

1. Usability: Should be user-friendly and only required detail should be shown in a minimal way
2. Reliability : The system should be user friendly to use.
3. Scalability: To increase the load of data handling
4. Flexibility : can run on any Platform

3.2.3 Sub- Systems

- Login

  Function: Used for login
  Input: User, Provider, Producer login Details
  Pre-Condition: None
  Source : From User, Provider, Producer
  Post-Condition: Redirected to Dashboard if Credentials are valid
  Exception: Error message is shown if credentials don't match
- Register
  Function: Used for registering account
  Input: User, Provider, Producer all details
  Pre-Condition: None
  Source : From User, Provider, Producer
  Post-Condition: Redirected to Login if details are valid and correct
  Exception: Error message is shown if any field is wrong or not valid

- Users-Info Update
  Function: Used for updating account
  Input: User, Provider, Producer all details to change
  Pre-Condition: None
  Source : From User, Provider, Producer
  Post-Condition: Users details are updated
  Exception: Error message is shown if any field is wrong or not valid

- Post-Review
  Function: Used for Posting Reviews
  Input: User Reviews
  Pre-Condition: None
  Source : From User

Post-Condition: Users reviews get added
Exception: Error message is shown if anything goes wrong

- Book appointment
  Function: Used for booking an appointment
  Input: User to book appointment
  Pre-Condition: None
  Source : From User
  Post-Condition: Appointment is booked
  Exception: Error message is shown if anything goes wrong

- Set appointments
  Function: Used for setting appointments
  Input: Provider, Producer all details for setting appointment
  Pre-Condition: None
  Source : Provider
  Post-Condition: Appointment is generated
  Exception: Error message is shown if anything goes wrong

- Buy Vaccines
  Function: Used for Buying vaccines
  Input: Provider all details for buying vaccines
  Pre-Condition: None
  Source : Provider
  Post-Condition: Order is Placed
  Exception: Error message is shown if anything goes wrong

- Update Vaccine Stocks
  Function: Used for updating vaccine stocks
  Input: Producer all details of vaccine
  Pre-Condition: None
  Source : Producer
  Post-Condition: Vaccine Stocks is updated or created
  Exception: Error message is shown if anything goes wrong

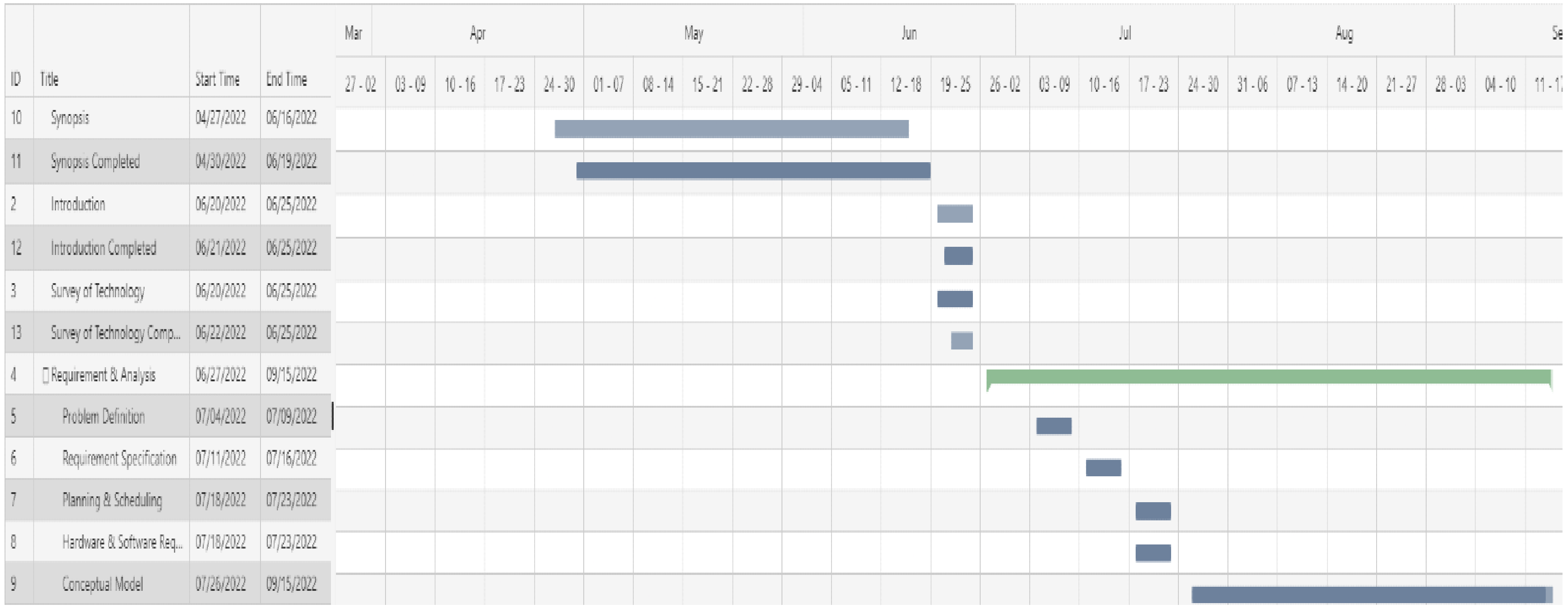
  //three more remaining

3.3 Planning & Scheduling

3.3.1 Activity Table

Activities	Start-Date	End-Date
Synopsis	27/04/22	16/06/22
1. Introduction	20/06/22	25/06/22
2. Survey of Technology	20/06/22	25/06/22
3. Requirement and analysis	27/06/22	15/09/22
4.1 Problem Definition	4/07/22	9/07/22
4.2 Requirement Specification	11/07/22	16/07/22
4.3 Planning & Scheduling	18/07/22	23/07/22
4.4 Hardware & Software Requirement	18.07/22	23/07/22
4.5 Conceptual Models	26/07/22	15/09/22

3.3.2 Gantt Chart



ID	Title	Start Time	End Time
10	Synopsis	04/27/2022	06/16/2022
11	Synopsis Completed	04/30/2022	06/19/2022
2	Introduction	06/20/2022	06/25/2022
12	Introduction Completed	06/21/2022	06/25/2022
3	Survey of Technology	06/20/2022	06/25/2022
13	Survey of Technology Comp...	06/22/2022	06/25/2022
4	Requirement & Analysis	06/27/2022	09/15/2022
5	Problem Definition	07/04/2022	07/09/2022
6	Requirement Specification	07/11/2022	07/16/2022
7	Planning & Scheduling	07/18/2022	07/23/2022
8	Hardware & Software Req...	07/18/2022	07/23/2022
9	Conceptual Model	07/26/2022	09/15/2022

Figure 3.3.1

3.4 Hardware & software requirements

3.5.1 Hardware

- Processor : Intel Core 3.0 2.3 Ghz or more.
- RAM : 4GB or more.
- Monitor: 17 CRT or LCD, Plasma etc.
- Hard-Disk: 256 or more (SSD preferable)
- Keyboard: Normal or multimedia.
- Mouse: Compatible

3.5.2 Software

- System O.S: Window or Linux (Debian or Arch).
- Frontend: HTML(ejs),JS,TailwindCss.
- Backend: Node.js,Express.js
- Database: MongoDb(NoSql)

3.5 Conceptual Model

3.5.1 Data Model

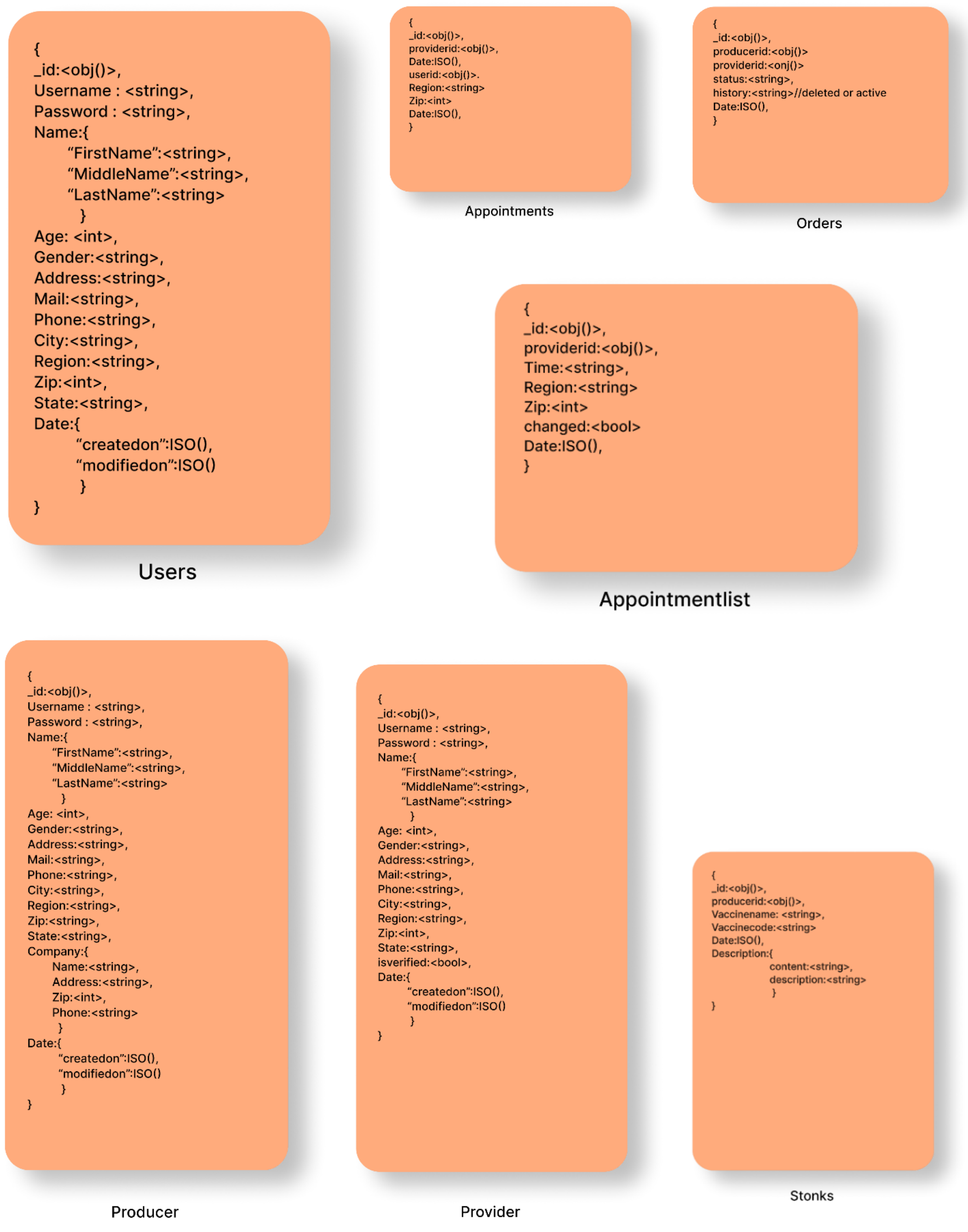
Unlike SQL databases, where you must determine and declare a table's schema before inserting data, MongoDB's collections, by default, do not require their documents to have the same schema. That is:

- The documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection.

Embedded Data

Embedded documents capture relationships between data by storing related data in a single document structure. MongoDB documents make it possible to embed document structures in a field or array within a document. These *denormalized* data models allow applications to retrieve and manipulate related data in a single database operation.

The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data. MongoDB allows related data to be embedded within a single document.



Users

```
{
_id:<obj()>,
Username : <string>,
Password : <string>,
Name:{
      "FirstName":<string>,
      "MiddleName":<string>,
      "LastName":<string>
        }
Age: <int>,
Gender:<string>,
Address:<string>,
Mail:<string>,
Phone:<string>,
City:<string>,
Region:<string>,
Zip:<int>,
State:<string>,
Date:{
        "createdon":ISO(),
        "modifiedon":ISO()
        }
}
```

Appointments

```
{
_id:<obj()>,
providerid:<obj()>,
Date:ISO(),
userid:<obj()>.
Region:<string>
Zip:<int>
Date:ISO(),
}
```

Orders

```
{
_id:<obj()>,
producerid:<obj()>
providerid:<onj()>
status:<string>,
history:<string>//deleted or active
Date:ISO(),
}
```

Appointmentlist

```
{
_id:<obj()>,
providerid:<obj()>,
Time:<string>,
Region:<string>
Zip:<int>
changed:<bool>
Date:ISO(),
}
```

Producer

```
{
_id:<obj()>,
Username : <string>,
Password : <string>,
Name:{
      "FirstName":<string>,
      "MiddleName":<string>,
      "LastName":<string>
        }
Age: <int>,
Gender:<string>,
Address:<string>,
Mail:<string>,
Phone:<string>,
City:<string>,
Region:<string>,
Zip:<string>,
State:<string>,
Company:{
      Name:<string>,
      Address:<string>,
      Zip:<int>,
      Phone:<string>
        }
Date:{
        "createdon":ISO(),
        "modifiedon":ISO()
        }
}
```

Provider

```
{
_id:<obj()>,
Username : <string>,
Password : <string>,
Name:{
      "FirstName":<string>,
      "MiddleName":<string>,
      "LastName":<string>
        }
Age: <int>,
Gender:<string>,
Address:<string>,
Mail:<string>,
Phone:<string>,
City:<string>,
Region:<string>,
Zip:<int>,
State:<string>,
isverified:<bool>,
Date:{
        "createdon":ISO(),
        "modifiedon":ISO()
        }
}
```

Stonks

```
{
_id:<obj()>,
producerid:<obj()>,
Vaccinename: <string>,
Vaccinecode:<string>
Date:ISO(),
Description:{
            content:<string>,
            description:<string>
             }
}
```

Figure 3..1

1. Data-Flow Diagram

A **data flow diagram** (or DFD) is a graphical representation of the information flow in a business process. It demonstrates how data is transferred from the input to the file storage and reports generation. By visualizing the system flow, the flow charts will give users helpful insights into the process and open up ways to define and improve their business.

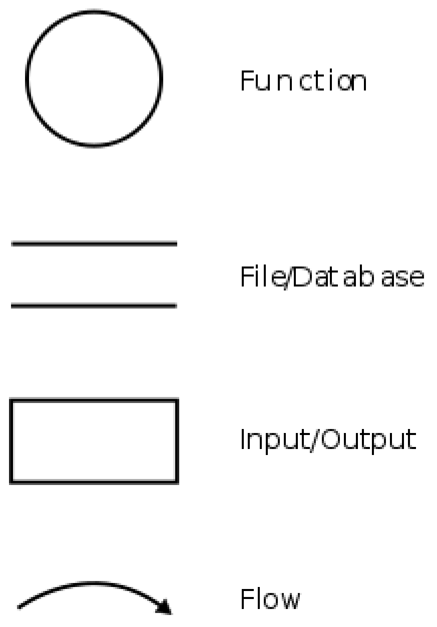


Figure 3..2

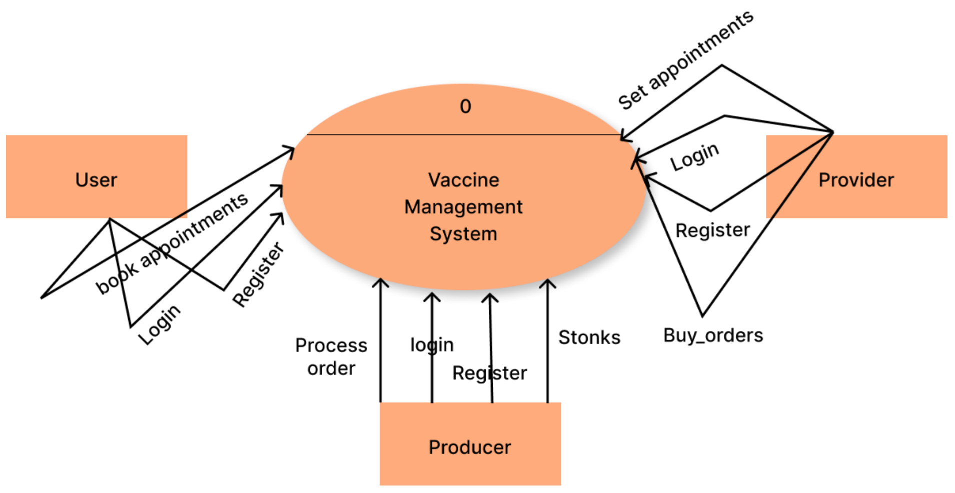


Figure 3.3

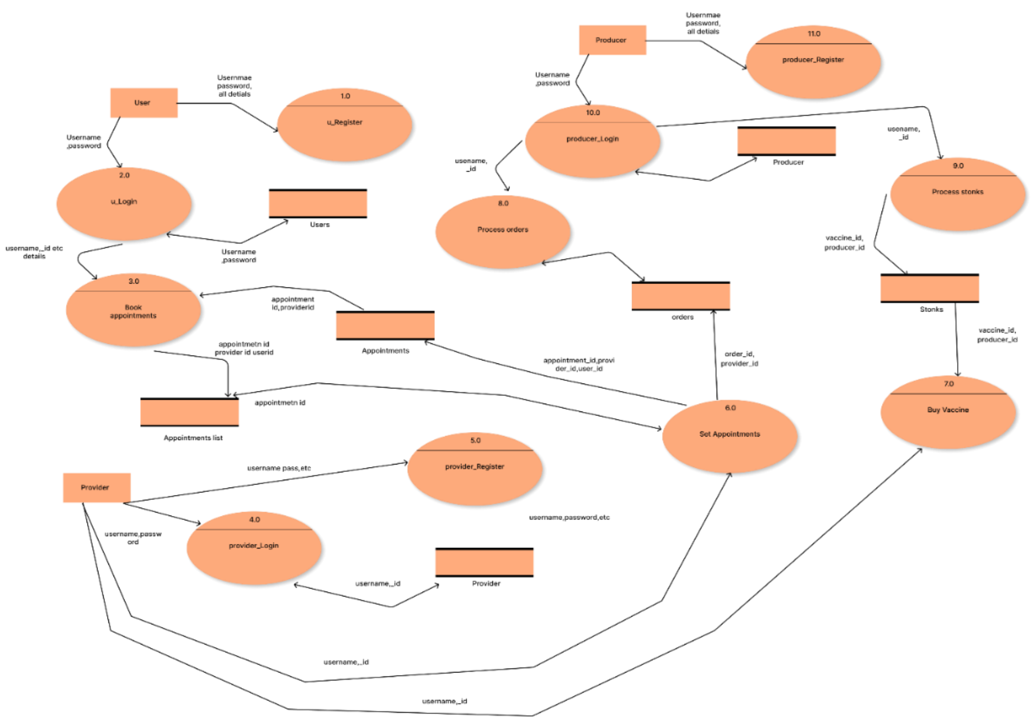


Figure 3.4

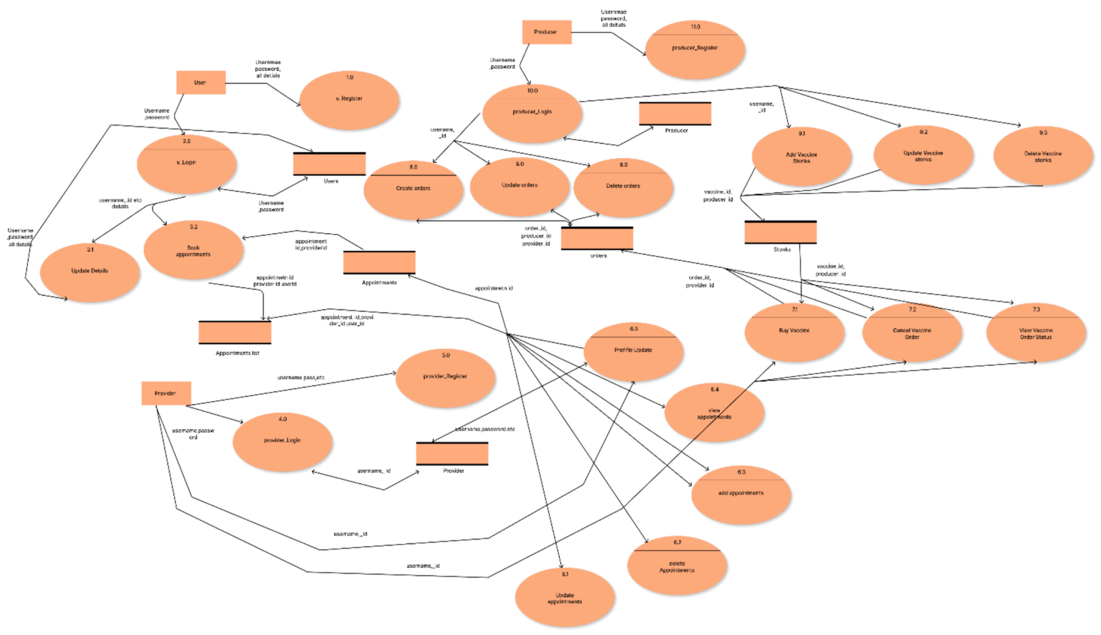

User
Usernmae
password,
all detials
1.0
u_Register
Username
,password
2.0
u_Login
Users
Username
,password
username,_id etc
details
Username
,password,
all details
3.1
Update Details
3.2
Book
appointments
appointment
id,providerid
Appointments
appointmetn id
provider id userid
Appointments list
appointment_id,provi
der_id,user_id
appointmetn id
Producer
Usernmae
password,
all detials
11.0
producer_Register
Username
,password
10.0
producer_Login
Producer
username,
_id
8.0
Create orders
8.0
Update orders
8.0
Delete orders
order_id,
producer_id
provider_id
orders
username,
_id
9.1
Add Vaccine
Stonks
9.2
Update Vaccine
stonks
9.3
Delete Vaccine
stonks
vaccine_id,
producer_id
Stonks
vaccine_id,
producer_id
order_id,
provider_id
7.1
Buy Vaccine
7.2
Cancel Vaccine
Order
7.3
View Vaccine
Order Status
6.5
Proffile Update
5.0
provider_Register
Provider
username pass,etc
usemame,passw
ord
4.0
provider_Login
username,password,etc
Provider
username,_id
6.4
view
appointments
6.3
add appointments
usaname,_id
username,_id
6.2
delete
Appointments
6.1
Update
appointments

Figure 3.5

3.5.3 Use-Case Diagram

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve
- The scope of your system

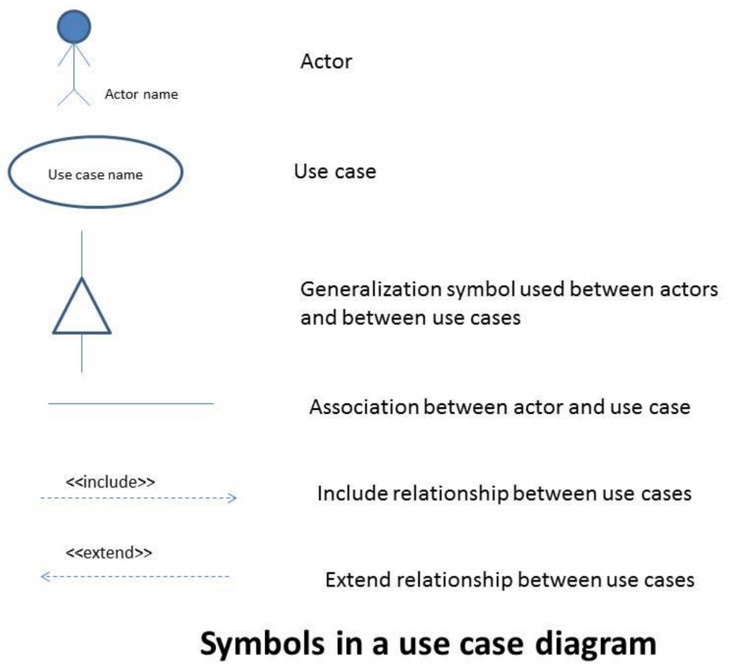

Actor name
Actor
Use case name
Use case
Generalization symbol used between actors and between use cases
Association between actor and use case
<<include>>
Include relationship between use cases
<<extend>>
Extend relationship between use cases
Symbols in a use case diagram

Figure 3.6

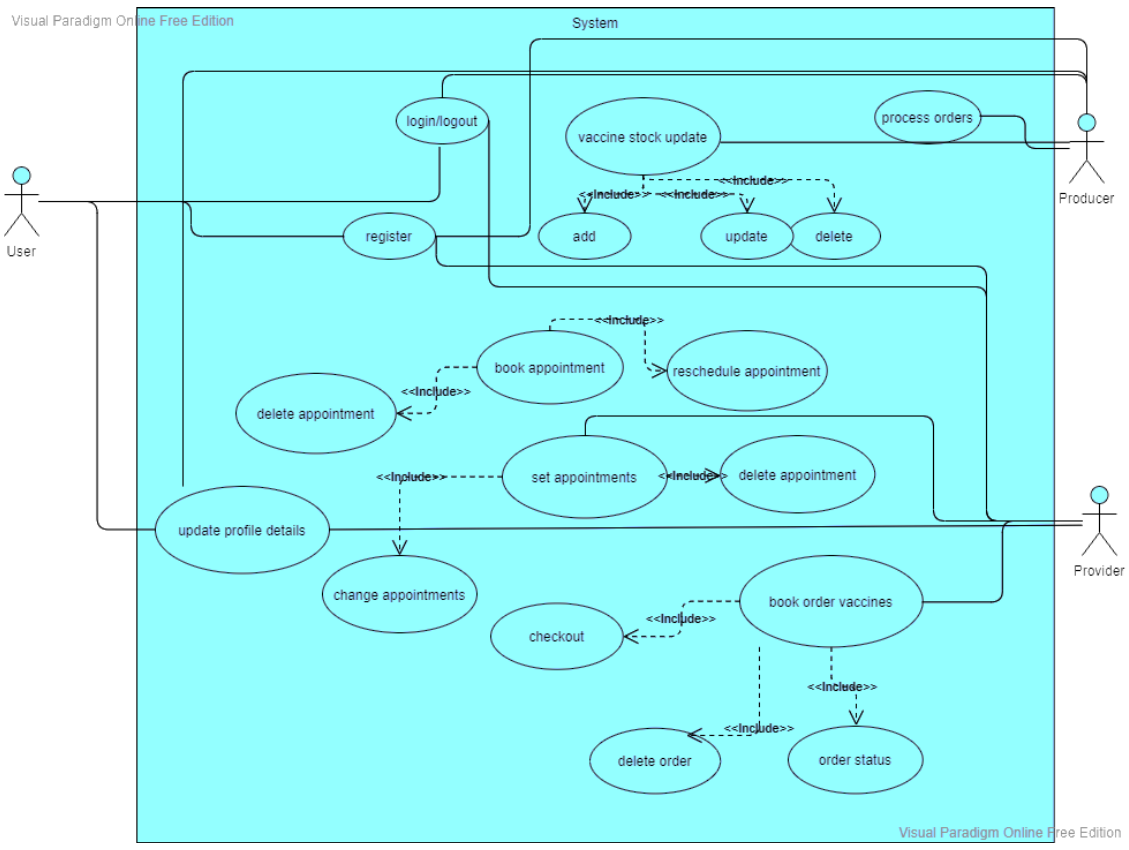


Figure 3.7

Use Case Description:

- Login

  Summary: Used to Login and Logout of the system

  Actor: User, Provider, Producer.

  Pre-condition: None.

  Description: The System waits for the credentials to be entered for validations and necessary for logout

  Exception: If credentials are not valid an error message is thrown

  Post-condition: Display user dashboard

- Register

  Summary: Used for registration

Actor: User, Provider, Producer.

Pre-condition: None.

Description: Is used for creating and account in the service by fill certain details

Exception: If proper fields are not valid an error message is thrown

Post-condition: redirect to login

- Book a appointment

  Summary: Used for booking an appointment

  Actor: User.

  Pre-condition: The user should be logged in with proper account details.

  Description: Is used for creating and account in the service by fill certain details

  Exception: If slots gets filled during registration error message is shown

  Post-condition: Appointment is booked.

- Book orders

  Summary: Used for buying vaccines.

  Actor: Provider.

  Pre-condition: Provider should be logged in and valid

  Description: For buying vaccines from a domestic producer

  Exception: If proper fields are not valid an error message is thrown

  Post-condition: Order is booked

- Set Appointments

  Summary: Used for setting up appointments

  Actor: Provider.

  Pre-condition: Provider should be logged in and valid

  Description: For setting appointments for users in his/her locality

  Exception: If proper fields are not valid an error message is thrown

  Post-condition: Appointment is successfully registered.

- Process Orders

  Summary: Used for processing orders.

Actor: Producer.

Pre-condition: None.

Description: For processing vaccines orders of a provider.

Exception: If an internal problem happens error message is thrown.

Post-condition: Order is successfully processed

3.5.4 Sequence Diagram

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

- Model high-level interaction between active objects in a system
- Model the interaction between object instances within a collaboration that realizes a use case
- Model the interaction between objects within a collaboration that realizes an operation
- Either model generic interactions (showing all possible paths through the interaction) or specific instances of a interaction (showing just one path through the interaction)

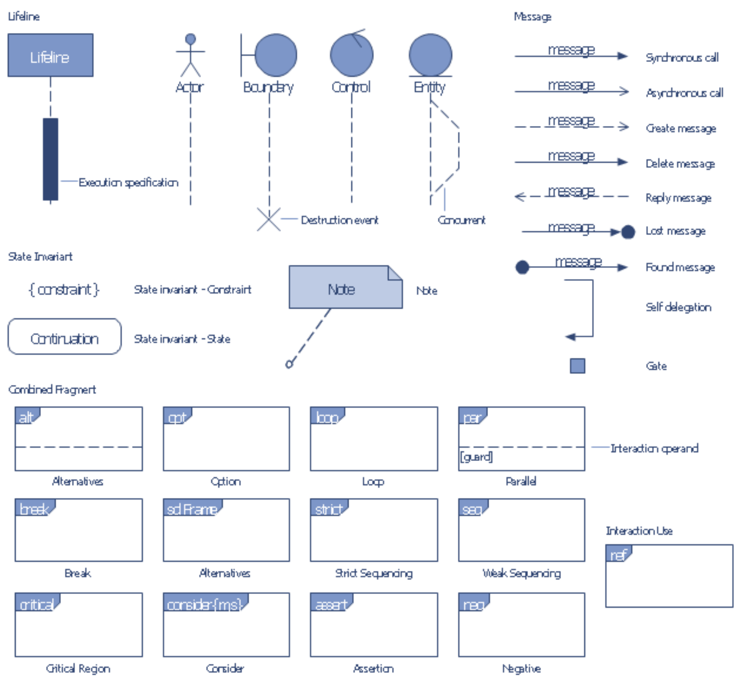
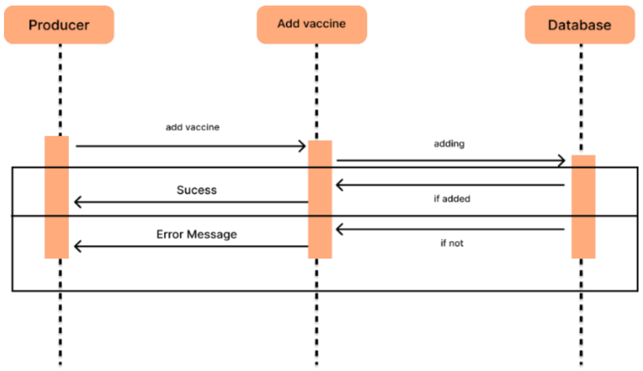
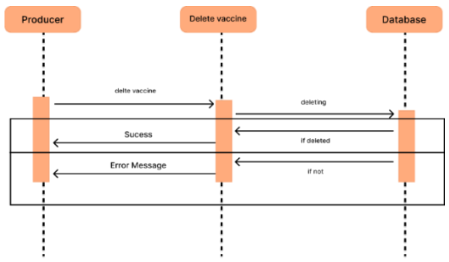


Figure 3.8



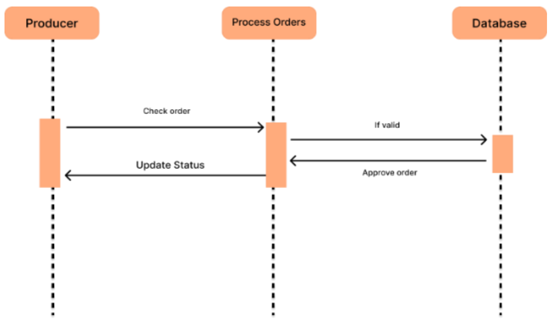
Add vaccine

Figure 3.9



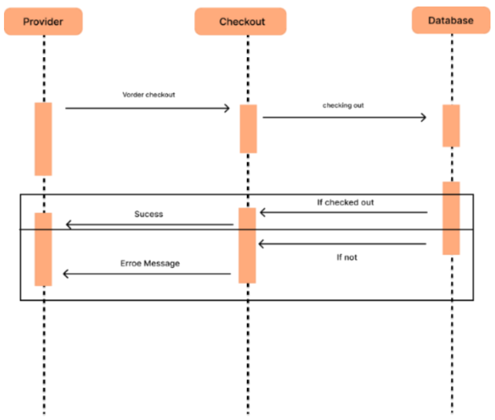
update vaccine

Figure 3.11



Process Orders

Figure 3.10



checkout

Figure 3.12

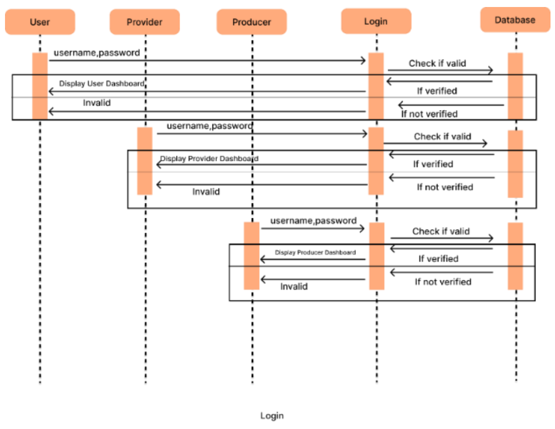


Figure 3.13

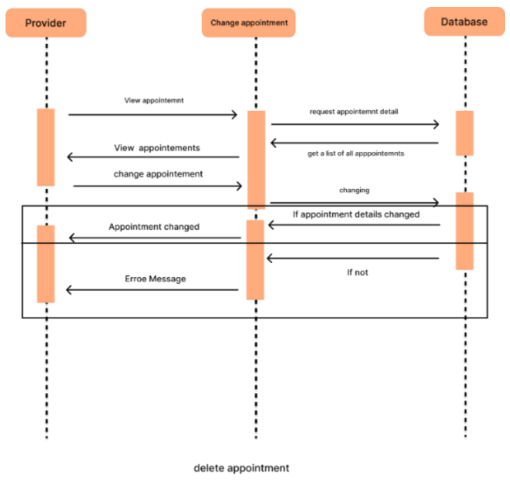


Figure 3.14

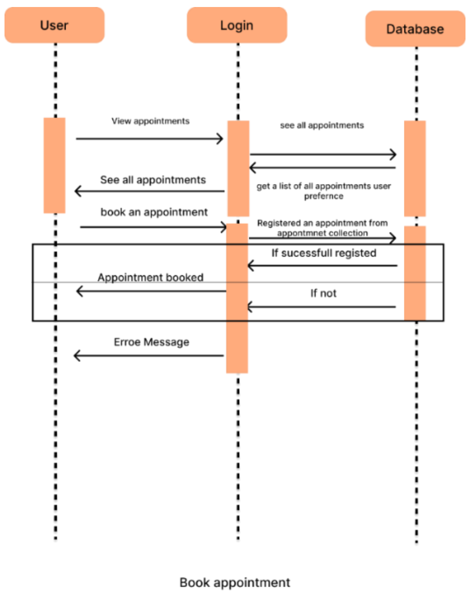


Figure 3.16

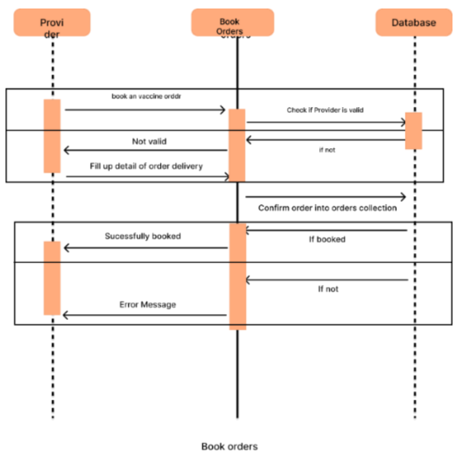


Figure 3.15

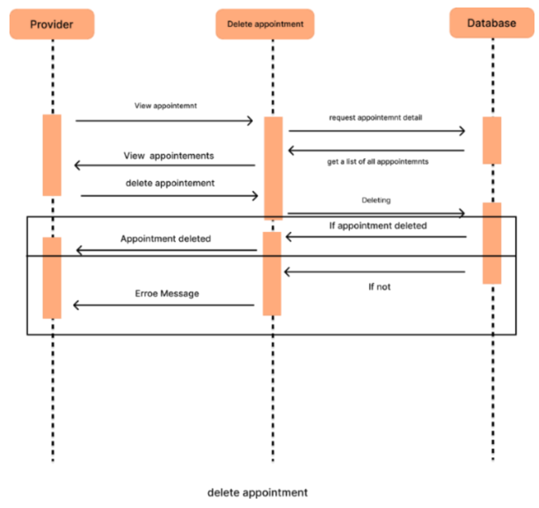


Figure 3.17

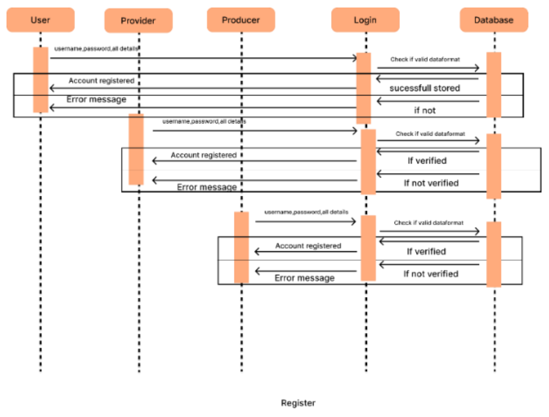


Figure 3.18

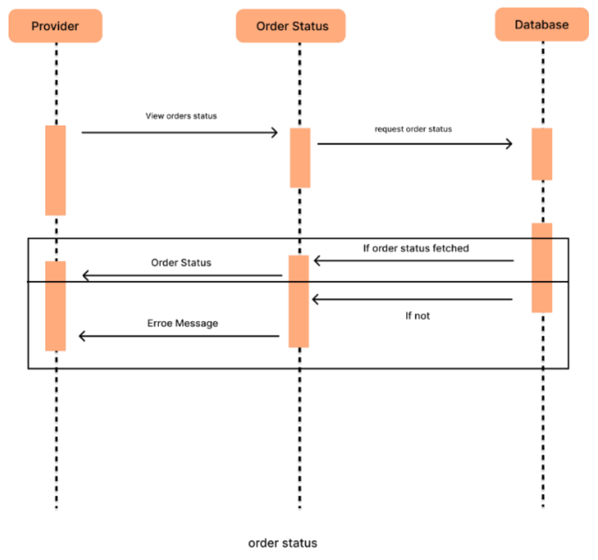
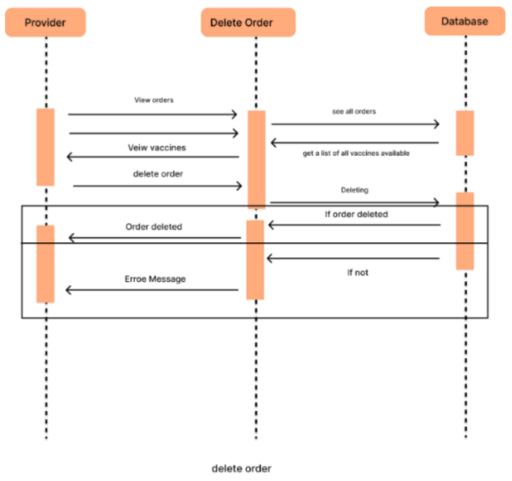
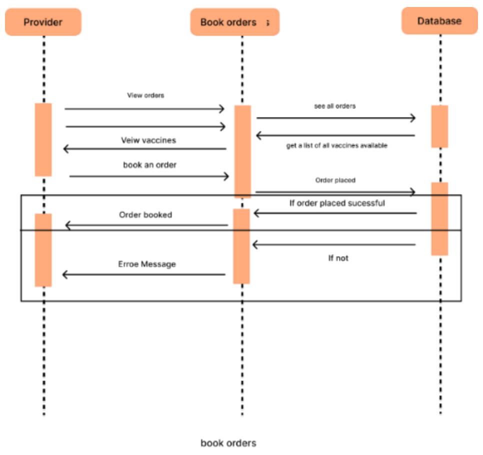


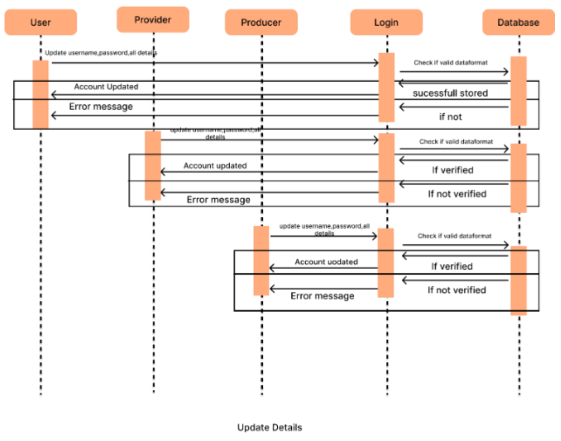
Figure 3.19



Fiqure 3.20



Fiqure 3.21



Fiqure 3.22

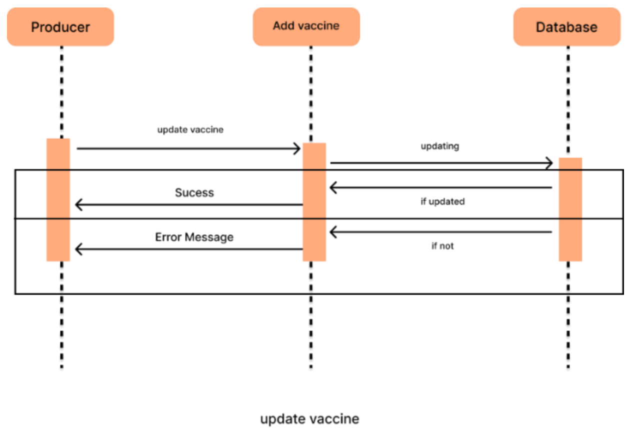


Figure 3.23

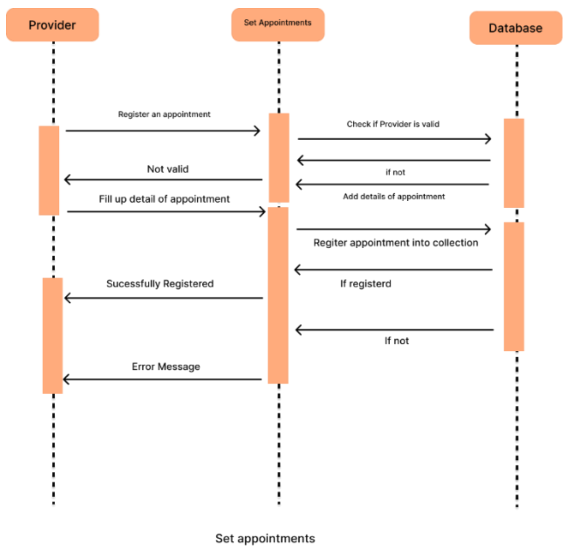


Figure 3.24

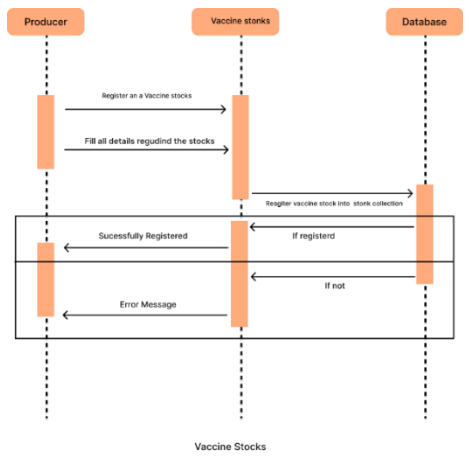

Producer
Vaccine stonks
Database
Register an a Vaccine stocks
Fill all details regudind the stocks
Resgiter vaccine stock into stonk collection
Sucessfully Registered
If registerd
If not
Error Message
Vaccine Stocks

Figure 3.25

3.5.5 Activity State Diagram

Activity diagrams present a number of benefits to users. Consider creating an activity diagram to:

- Demonstrate the logic of an algorithm.
- Describe the steps performed in a UML use case.
- Illustrate a business process or workflow between users and the system.
- Simplify and improve any process by clarifying complicated use cases.
- Model software architecture elements, such as method, function, and operation.

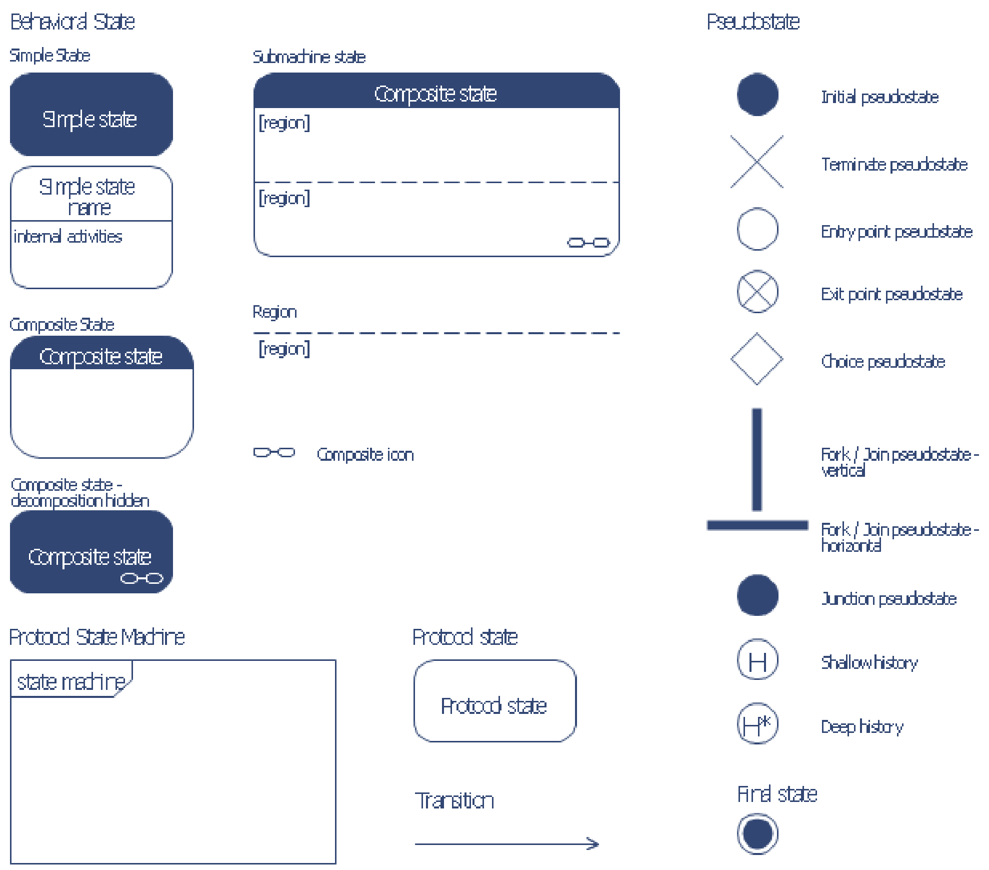


Figure 3.26

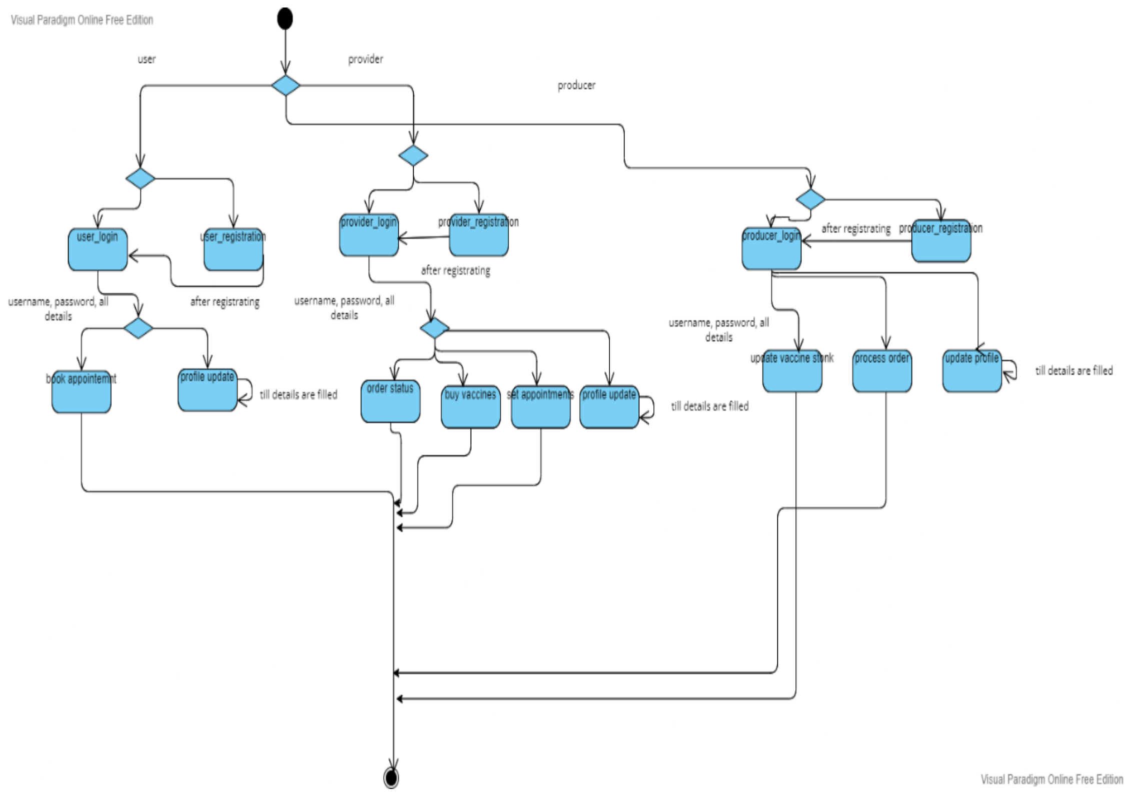


Figure 3.27

3.5.6 Activity Diagram

Activity diagrams are mainly used as a flowchart that consists of activities performed by the system. Activity diagrams are not exactly flowcharts as they have some additional capabilities. These additional capabilities include branching, parallel flow, swim-lane, etc.

Before drawing an activity diagram, we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities, we need to understand how they are associated with constraints and conditions.

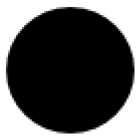
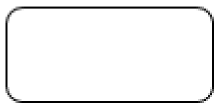
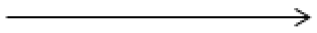
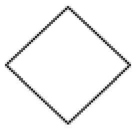
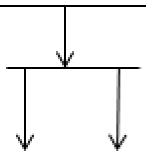
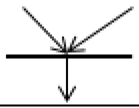
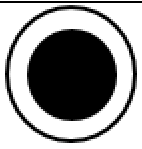
Sr. No	Name	Symbol
1.	Start Node	
2.	Action State	
3.	Control Flow	
4.	Decision Node	
5.	Fork	
6.	Join	
7.	End State	

Figure 3.28

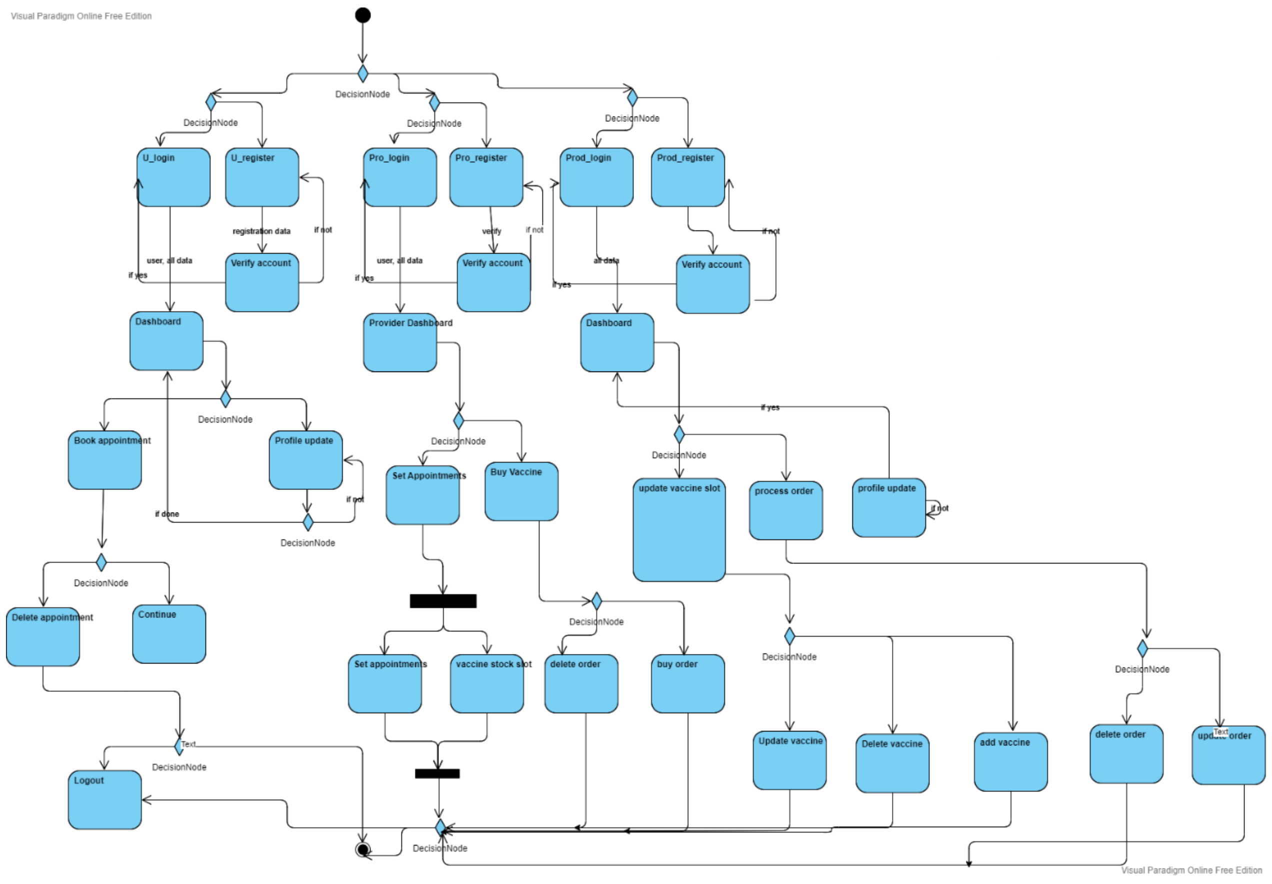

Visual Paradigm Online Free Edition
DecisionNode
U_login
U_register
Pro_login
Pro_register
Prod_login
Prod_register
registration data
if not
user, all data
if yes
Verify account
verify
Dashboard
Provider Dashboard
Book appointment
Profile update
Set Appointments
Buy Vaccine
update vaccine slot
process order
profile update
if done
Delete appointment
Continue
Set appointments
vaccine stock slot
delete order
buy order
Update vaccine
Delete vaccine
add vaccine
update order
Logout
Text

Figure 3.29

4. System Design

4.1 Interface Design

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna.

Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci.

Aenean nec lorem. In porttitor. Donec laoreet nonummy augue.

Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy.

4.2 Test Cases

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna.

Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci.

Aenean nec lorem. In porttitor. Donec laoreet nonummy augue.

Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy.

Bibliography

websites

- https://www.servicenow.com/solutions/vaccine-management.htm
- https://www2.deloitte.com/us/en/pages/public-sector/solutions/vaccine-management-system.htm
- https://www.intelex.com/products/applications/vaccine-management}{Intelex Vaccine Solution
- https://www.zenflowchart.com/guides/data-flow-diagram

Reference books

- Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna.
- Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus.
- Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci.
- Aenean nec lorem. In porttitor. Donec laoreet nonummy augue.
- Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy.