COMSATS University Islamabad,
Park Road, Chak Shahzad, Islamabad Pakistan

SOFTWARE DESIGN DESCRIPTION

(SDD DOCUMENT)

for

Child Immunization and Growth Tracking System

Version 1.0

By

Shariq Ahmed **CIIT/SP18-BCS-151/ISB**

Umar Khalid **CIIT/SP18-BCS-164/ISB**

Supervisor

Dr. Ashfaq Hussain Farooqi

Bachelor of Science in Computer Science (2018-2022)

Table of Contents

Contents

1. Introduction 1
2. Design methodology and software process model 1
2.1.1 Process Methodology 1
2.1.2 Design Methodology 1
3. System overview 2
3.1 Architectural design 2
3.1.1 3 tier Architecture 2
3.2 Three Tier Architecture Description 3
3.2.1 Tier-1 3
3.2.2 Tier-2: 3
3.2.3 Tier-3: 3
3.3 Process flow/Representation 3
3.3.1 Activity Diagram for Hospital Module 3
3.3.2 2nd Activity Diagram for Hospital Module 5
3.3.3 Activity Diagram for Admin 6
3.3.4 2nd Acitivity Diagram For Admin 7
3.3.5 Activity Diagram for Vaccine Center Module 8
3.3.6 Activity Diagram For Child Growth And Parent Module 9
3.3.7 Activity Diagram For Polio Worker Module 10
3.3.8 2nd Activity Diagram for Vaccine Center Module 11
4. Design models 12
4.1 Class Diagram 12
4.1.1 Class Diagram For web 12
4.1.2 Class Diagram for Mobile 13
4.2 Sequence Diagram 13
4.2.1 Sequence Diagram For Add Child Data 14
4.2.2 Sequence Diagram For Campaign Creation 15
4.2.3 Sequence Diagram For Child Growth Prediction 16
4.2.4 Sequence Diagram For Child Vaccine Update 17
4.2.5 Sequence Diagram For Predict Vaccine Stock 18
4.2.6 Sequence Diagram For Polio Vaccine update and Symptoms Check 19
5. Data design 20
5.1 ERD 20
5.2 Data dictionary 20
5.2.1 Parent Schema 20
5.2.2 Vaccine Schema 21
5.2.3 User Schema 21
5.2.4 Polio Worker Schema 21
5.2.5 Vaccine Schema 22
5.2.6 Hospital Schema 22
5.2.7 Child Schema 23
5.2.8 Campaign Schema 23
6. Algorithm & Implementation 23
6.1 Implementation 23
6.1.1 Server 23
6.1.2 Campaign Routing 24
6.1.3 Child Routing 25
6.1.4 Vaccine Routing 27

7. Software requirements traceability matrix 28
8. Human interface design 31
8.1 Screen images 31

Table of Figures

Figure 1: 3- Tier Architecture 2
Figure 2: Hospital Activity 1 4
Figure 3: Hospital Activity 2 5
Figure 4: Admin Activity 1 6
Figure 5: Admin Activity 2 7
Figure 6: Vaccine Center Activity 1 8
Figure 7: Parent Activity 1 9
Figure 8:Polio Worker Activity 10
Figure 9: Vaccine Center Activity 11
Figure 10: Class Diagram for Web 12
Figure 11: Class Diagram For Mobile 13
Figure 12: Sequence Diagram For Add Child Data 14
Figure 13: Sequence Diagram For Campaign Creation 15
Figure 14: Sequence Diagram For Child Growth Prediction 16
Figure 15: Sequence Diagram For Child Vaccine Update 17
Figure 16: Sequence Diagram For Predict Vaccine Stock 18
Figure 17: Sequence Diagram For Polio Vaccine Update 19
Figure 18: ERD 20
Figure 19: Screen 1 32
Figure 20: Screen 2 33
Figure 21: Screen 3 34
Figure 22: Screen 4 35
Figure 23: Screen 5 35

Revision History

Name	Date	Reason for changes	Version

Application Evaluation History

Comments (by committee) ***include the ones given at scope time both in doc and presentation**	**Action Taken**
From where the data about children's and polio teams will be taken. Who will be the owner/Stakeholder of the project that will manage it ? Info about vaccines ? Module 7 will be a building a Data Warehouse for DSS ? Where is the use of AI ?	Children data will be taken by my friend's hospital Stakeholders can be government body or any organization under govt. Info about vaccines means details information about vaccine like whats the purpose of this vaccine, why we use this vaccine etc. No, we wont build data warehouse. We will not have that much data for a data warehouse. AI is used in future prediction models
The comments previously given is not entertained by the students. Must ensure building data warehouse for decision making system and Where and how to implement AI in the project.	On our communication with Sir Qasim Malik and research, we found out that Data warehouse is not needed in the project as our data is only coming from single source. For DDS, we'll apply algorithms on the data coming from website or mobile. For AI implementation, we are building a machine learning model, in which we will add children data regarding growth and skills. Upon that data model will predict about growth. On those results, diets will be suggested.

Supervised by

Dr. Ashfaq Hussain Farooqi

Signature_____

1. Introduction

Child immunization and growth tracking is a system designed to automate the current vaccination system with addition to track the child of growth. Mainly this system is performing these major tasks, i.e., centralizing data of children and parents, managing the vaccination process, growth tracking of child. It will help the authorities to prevent the diseases which are extinct in all over the world. Additionally, it can help the parents to keep an eye on their child health. Following are the modules our system has.

1. Centralizing data
2. Vaccine Center Management
3. User Management
4. Growth Tracking
5. Decision Support System

In this document we are going to explain all the basic structure and designs on which this project will be based on. Following are the points which we have implemented in our 40% submission.

- Front End of all the following points are implemented
- Database is structured and implemented
- Hospital: Centralizing of all data is done which means data of each new child can be enter on the time of birth. Children data can be search by his id or parent and can be viewed.
- Vaccine Center: Vaccine Centers can add or update their stocks. Apart from that, vaccine centers can make new vaccine campaigns and they can allot worker and vaccines to campaigns. Apart from that, they can also monitor their status.

2. Design methodology and software process model

2.1.1 Process Methodology

We are going to use the incremental model in our project. Reasons for choosing an incremental model are

- Our project is long which consists of different modules
- Some modules can be implemented individually
- Requirements of the project are defined and won't change during development
- So the incremental model suits our project as we can implement different modules and later we can develop and test them.
- In this way, the system will get a test in every possible way and we will get efficient results.

2.1.2 Design Methodology

We are going to Object Oriented Methodology. Reason for using OOP methodology is

- System can be decomposed in smaller parts components which makes management easy
- Communication between components become easy
- System becomes understandable and less complex which makes implementation easy
- System can be easily modified if required

3. System overview

Child Vaccination and growth has been a major problem of our health care system. Looking at the stats we come to know that we are only one of 2 countries who are having polio problem. Reason is that data is not centralized and its difficult to track vaccination and growth manually. In rural areas, some uneducated people thinks these vaccinations are dangerous for their child and that's why they don't get their children vaccinated. Due to this mortality rate increases. 66 out of 1000 children dies every year and major causes of these stats and unvaccinated and untracked growth of children

Child immunization will help the authorities to centralize the children data. And later they can track the vaccination automatically. On birth of child, his data will be entered in the system that will generate vaccination dates. Before vaccination date, parent will get reminder messages. If any children don't get vaccinated they will get reminder messages, and later on unvaccinated child data will be send to higher authorities. In this way it will help to track the children who are still unvaccinated.
Growth module is for parents to keep check on the health of their children. They can enter data into the system time to time and system on basis of data will give results on it. They can see suggested diet too.

3.1 Architectural design

3.1.1 3 tier Architecture

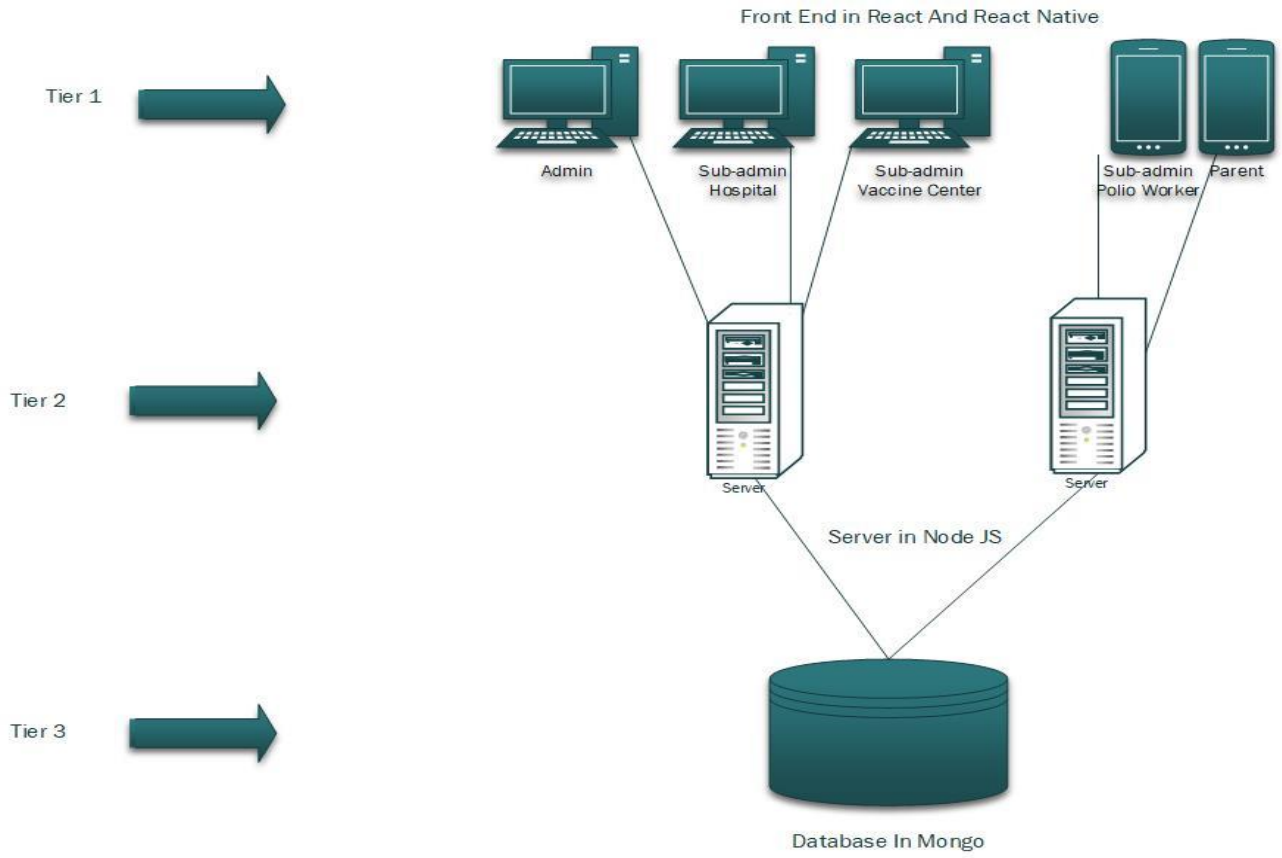

Figure 1: 3- Tier Architecture

3.2 Three Tier Architecture Description

3.2.1 **Tier-1**

This is the front end of the system where user can access all the functionalities using the interface. It is usually called as Presentation Layer. This layer is implemented in React and React Native.

3.2.2 **Tier-2:**

This is the backend of the system where all the application logic is implemented. It is a server which make communication between tier 1 and tier 3. This layer is called as Business Layer. It is implemented in Node JS.

3.2.3 Tier-3:

This is the database of the system where all the application data will be stored. This layer is called as Data Layer. It is implemented Mongo DB.

3.3 Process flow/Representation

3.3.1 Activity Diagram for Hospital Module

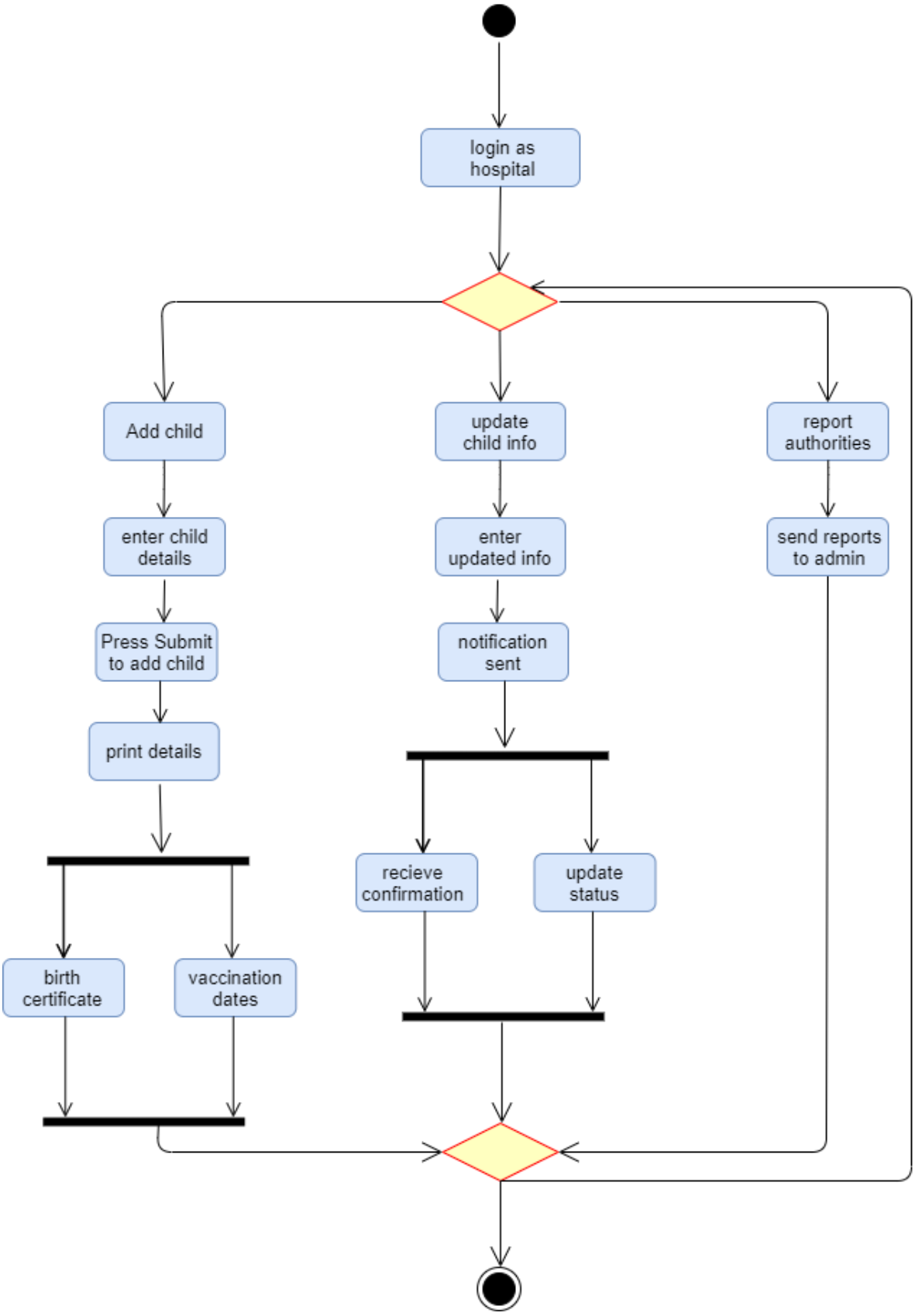

Figure 2: Hospital Activity 1

3.3.2 2nd Activity Diagram for Hospital Module

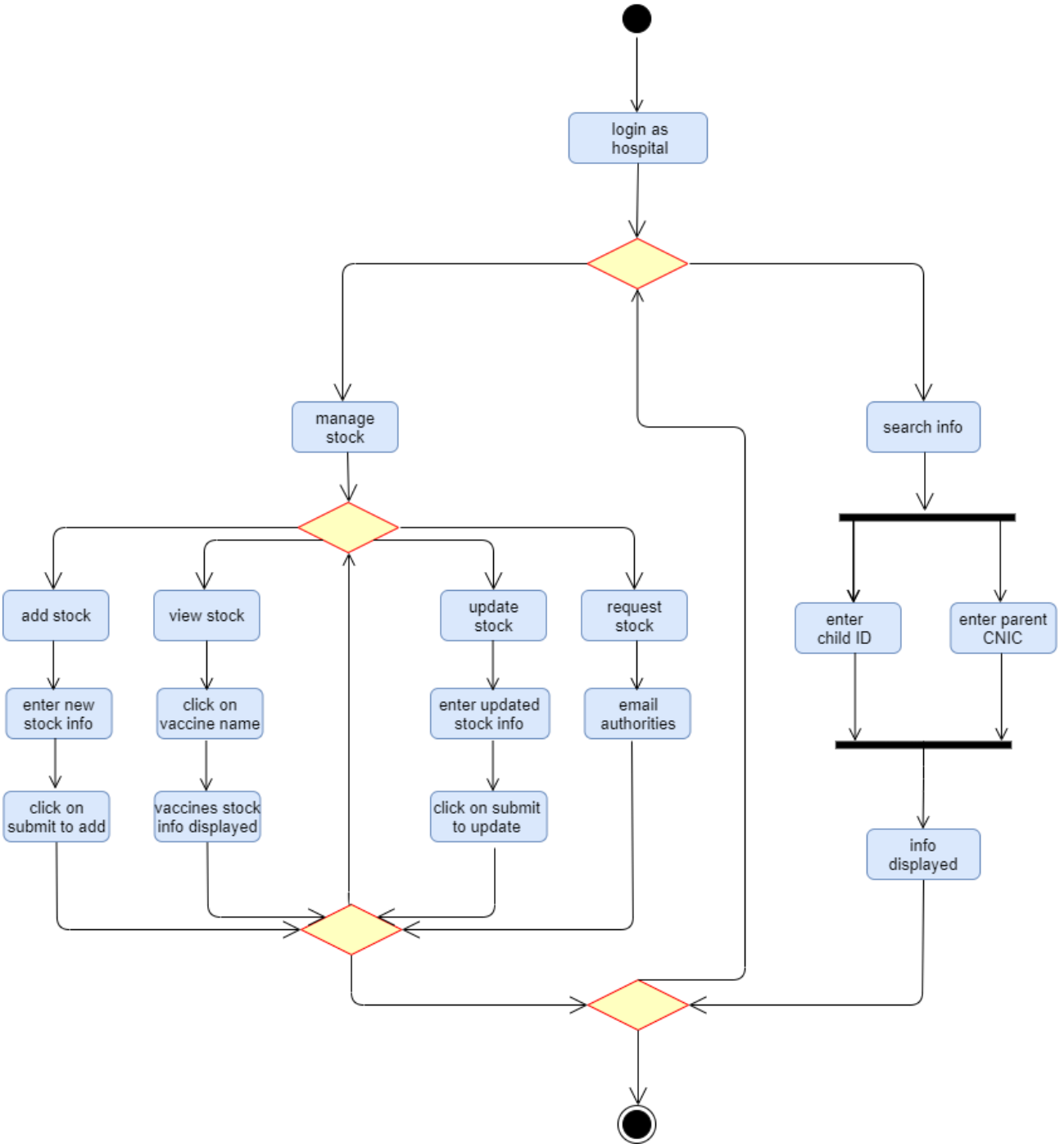

Figure 3: Hospital Activity 2

3.3.3 Activity Diagram for Admin

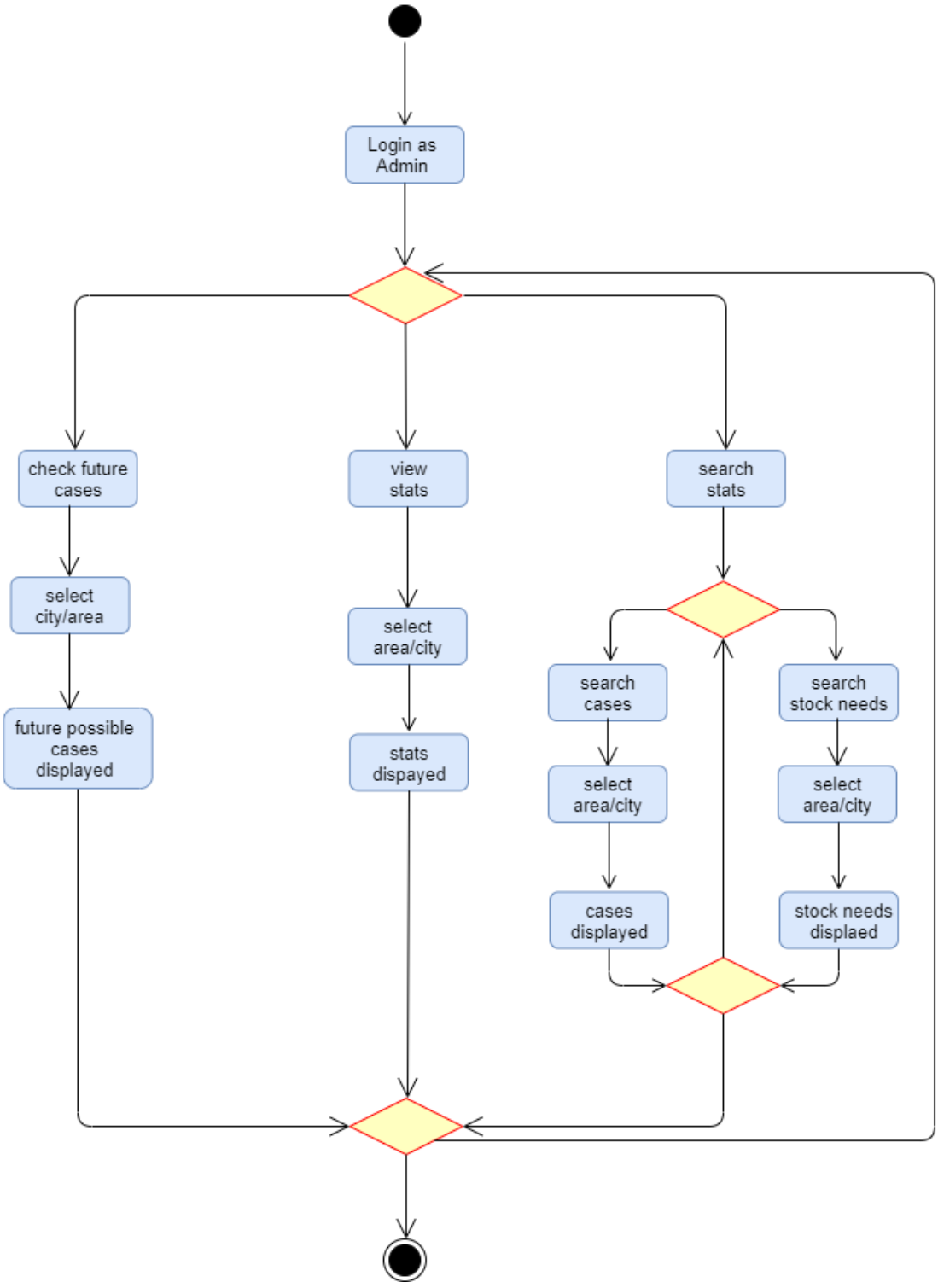

Figure 4: Admin Activity 1

3.3.4 2nd Acitivity Diagram For Admin

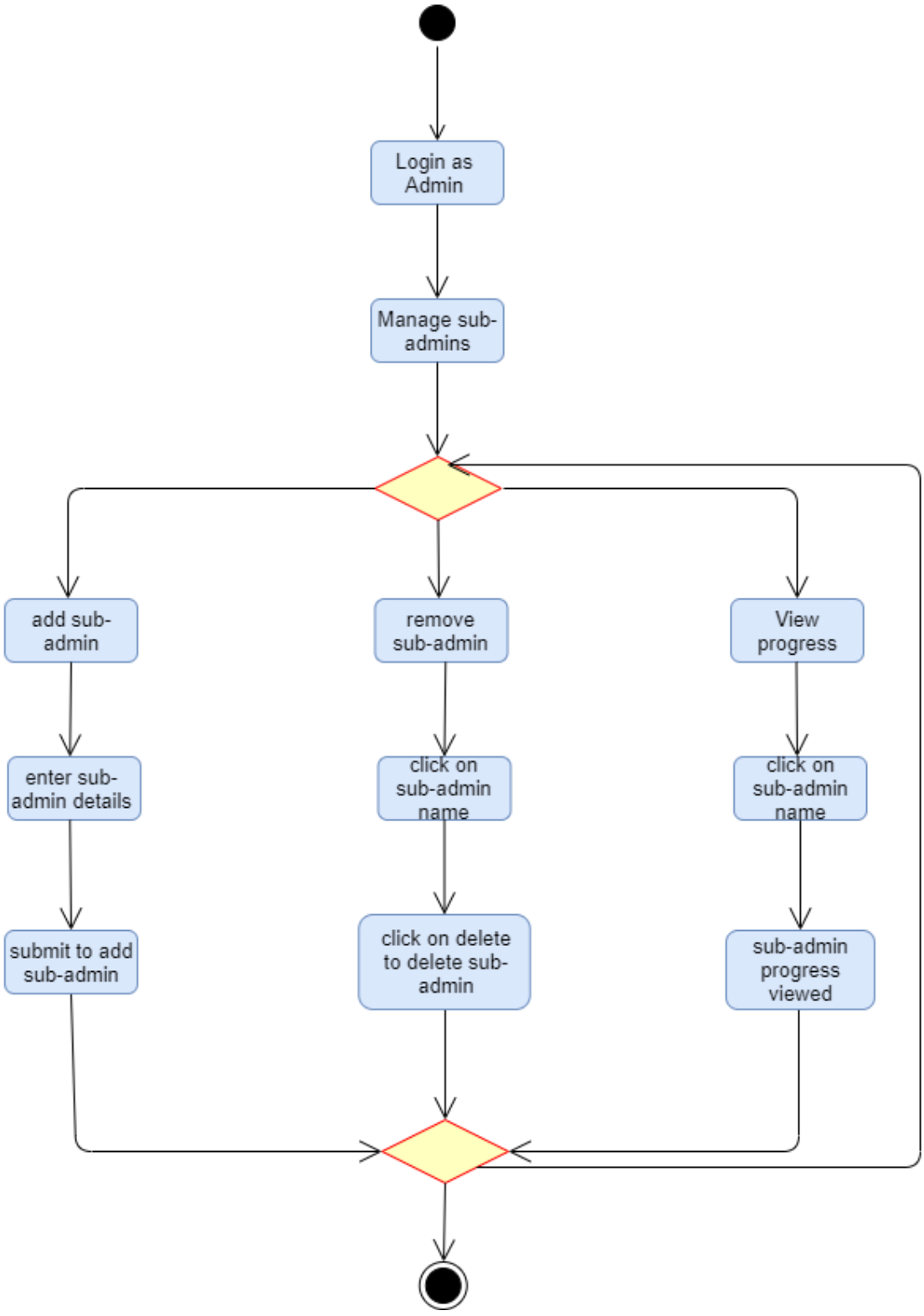

Figure 5: Admin Activity 2

3.3.5 Activity Diagram for Vaccine Center Module

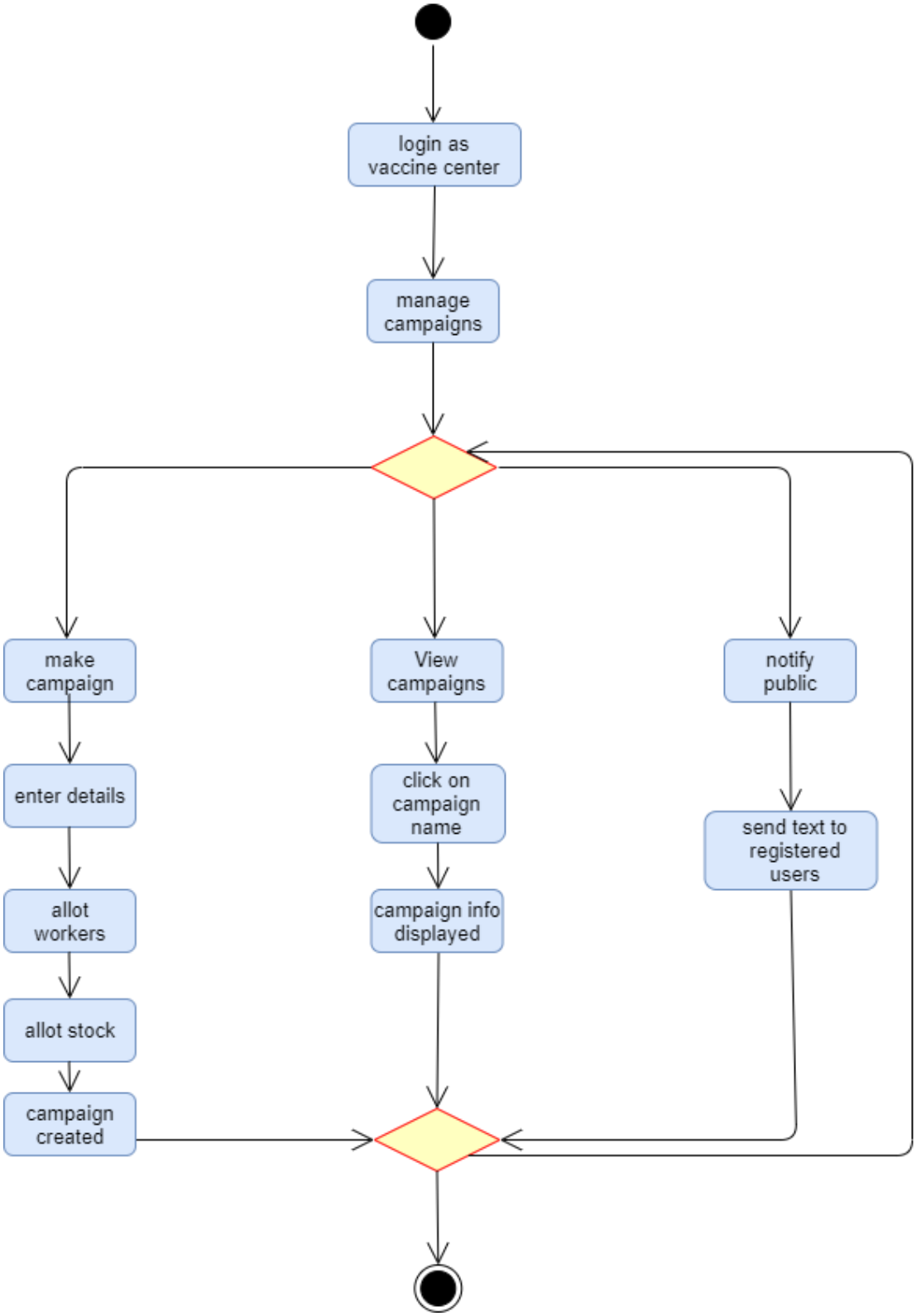

Figure 6: Vaccine Center Activity 1

3.3.6 Activity Diagram For Child Growth And Parent Module

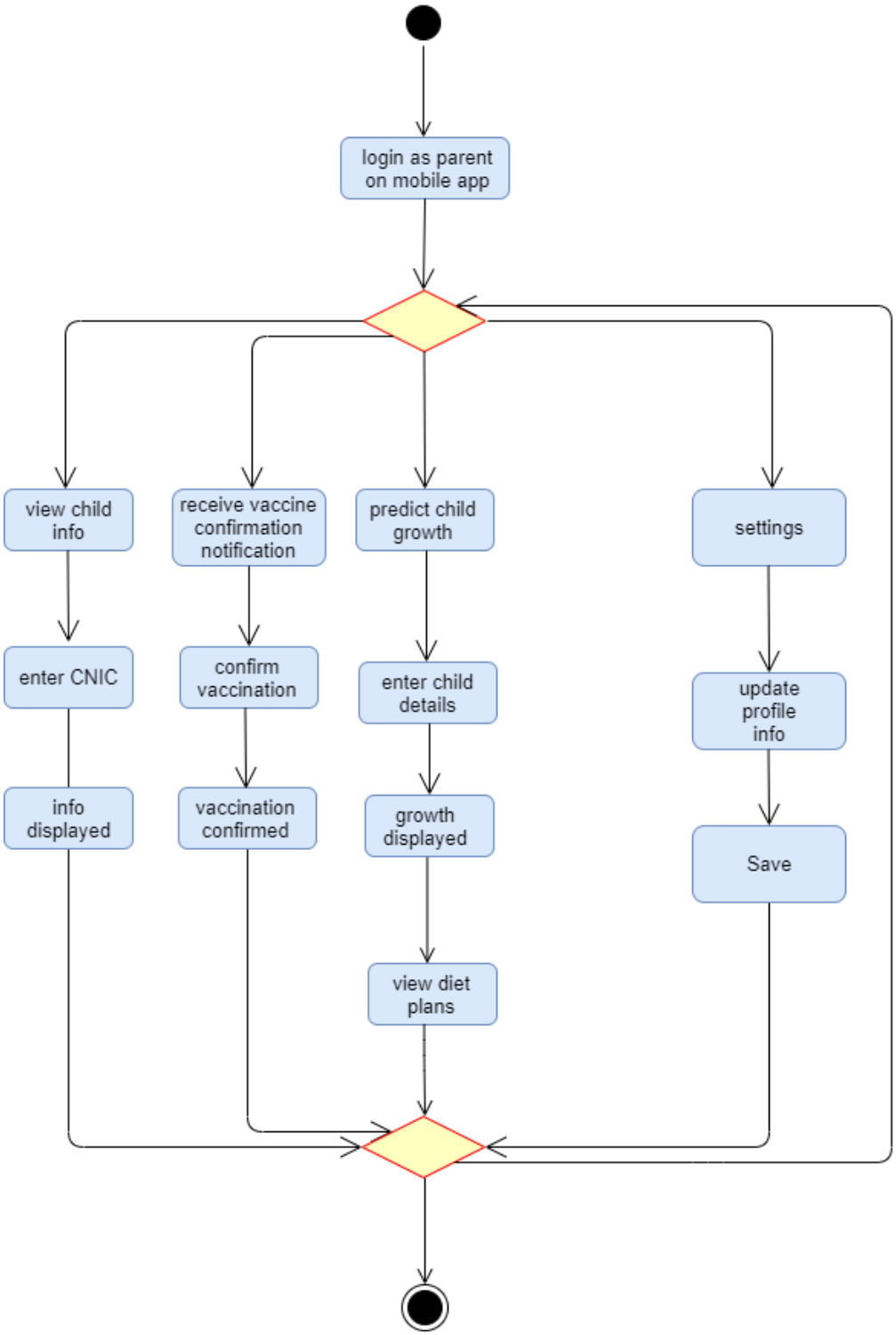

Figure 7: Parent Activity 1

3.3.7 Activity Diagram For Polio Worker Module

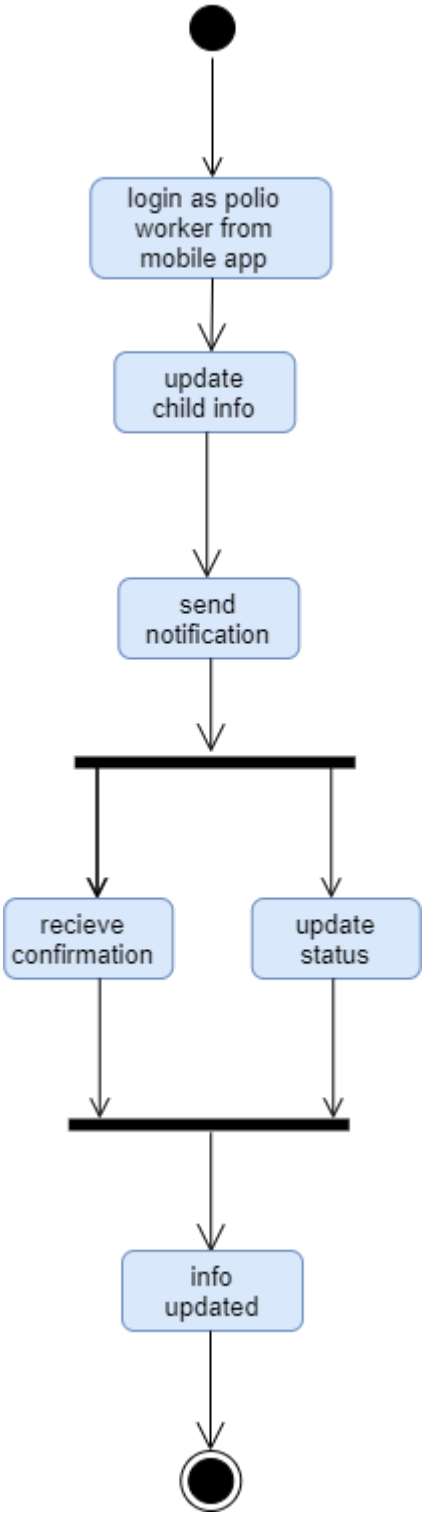

Figure 8:Polio Worker Activity

3.3.8 2nd Activity Diagram for Vaccine Center Module

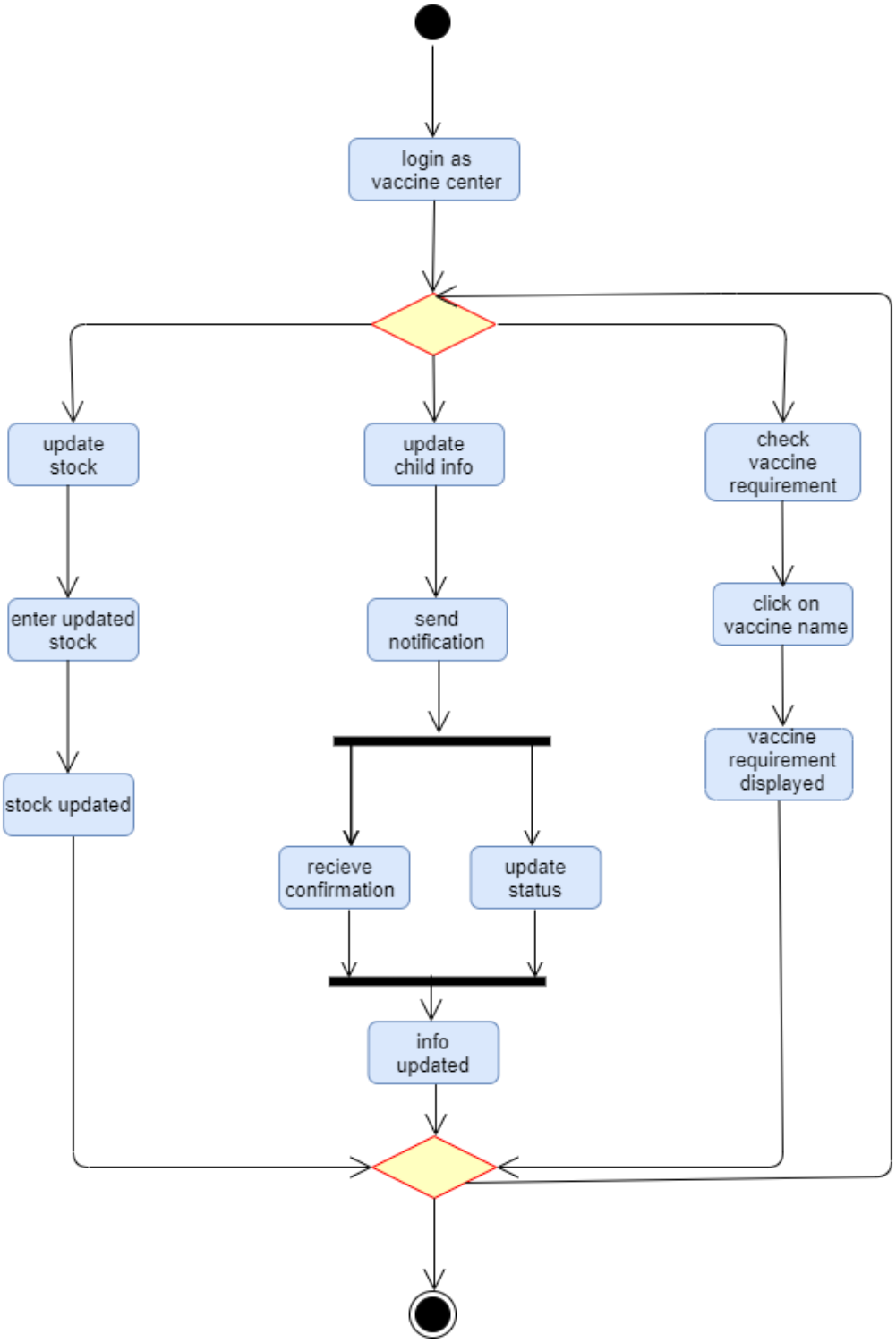

Figure 9: Vaccine Center Activity

4. Design models

4.1 Class Diagram

4.1.1 Class Diagram For web

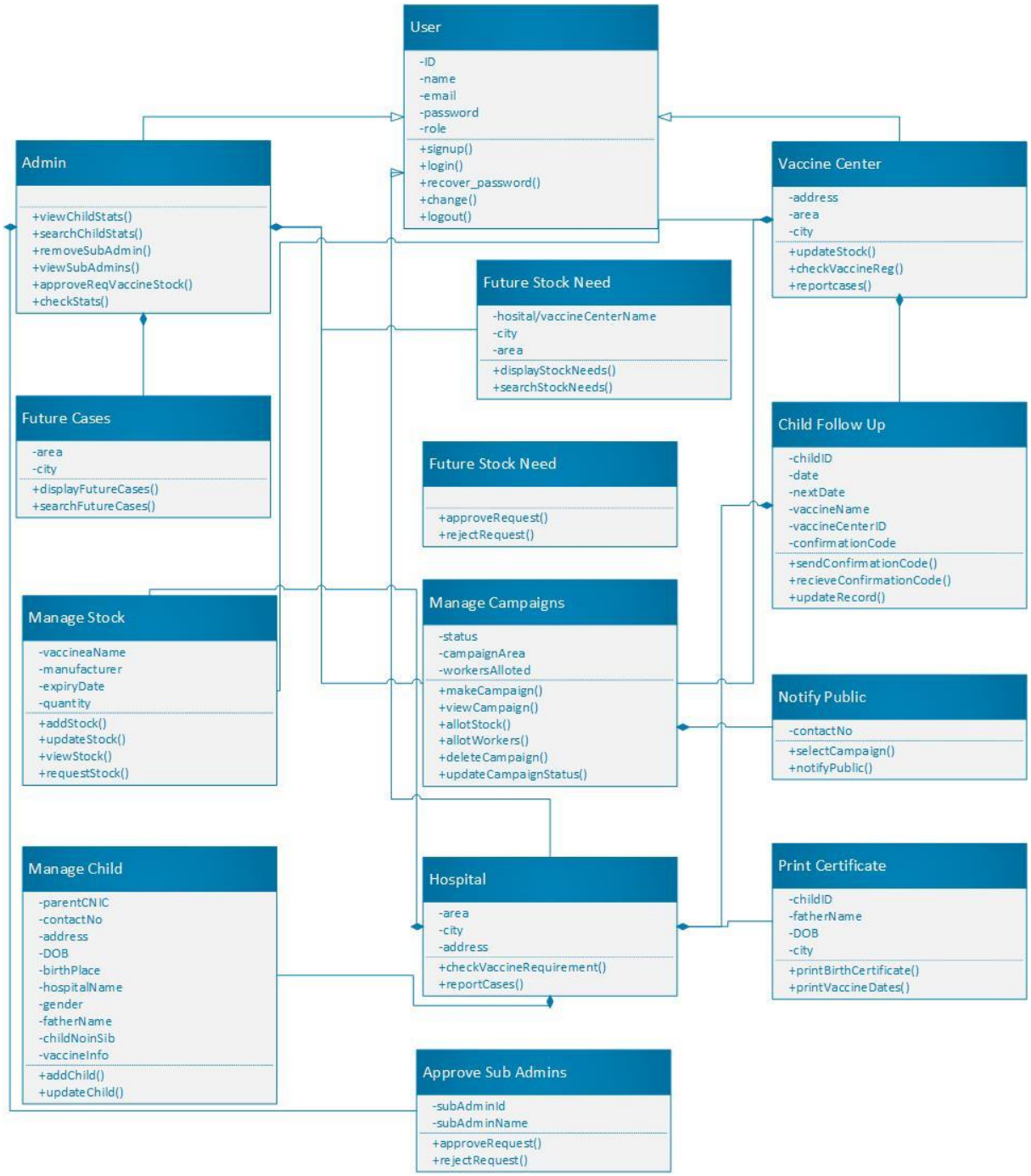

Figure 10: Class Diagram for Web

4.1.2 Class Diagram for Mobile

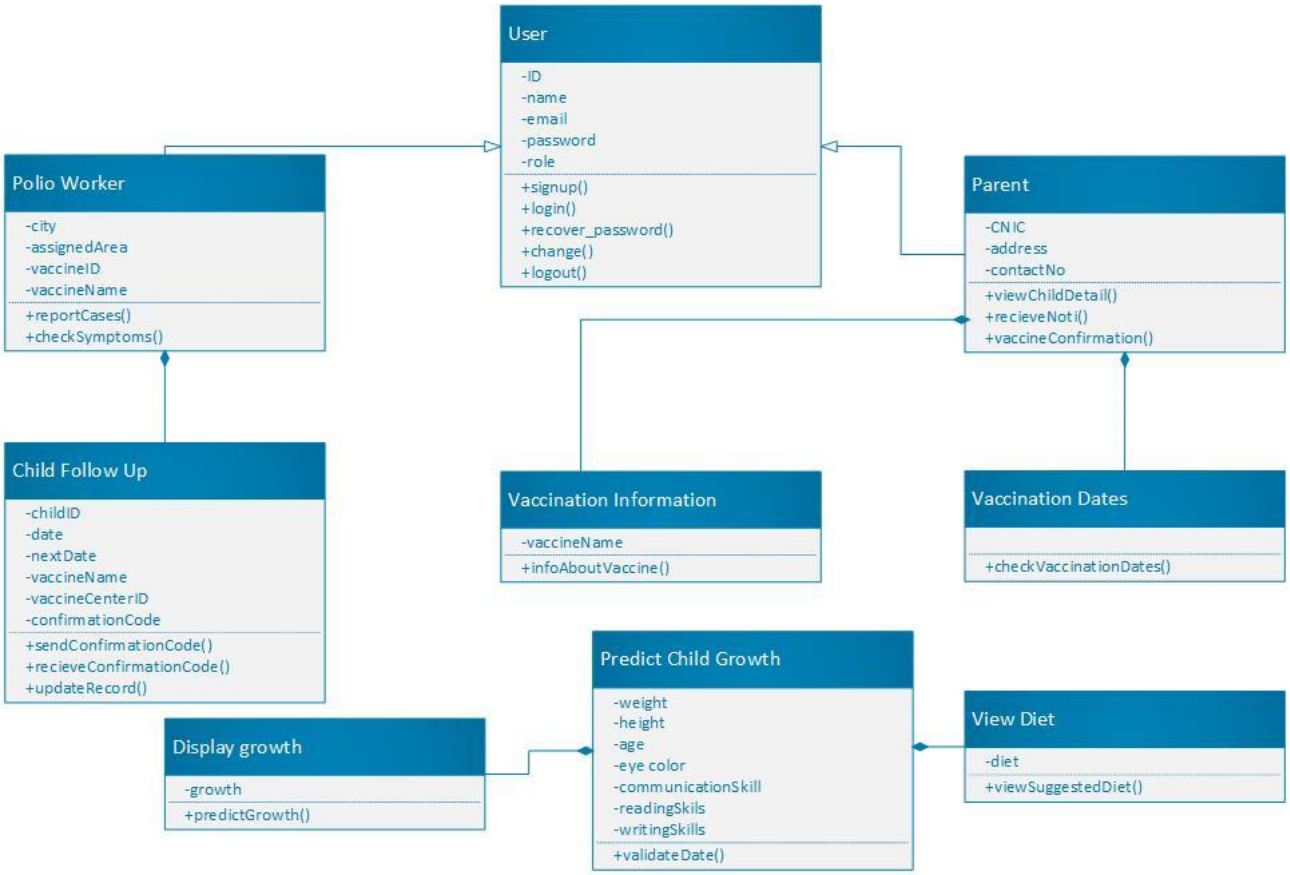

Figure 11: Class Diagram For Mobile

4.2 Sequence Diagram

4.2.1 Sequence Diagram For Add Child Data

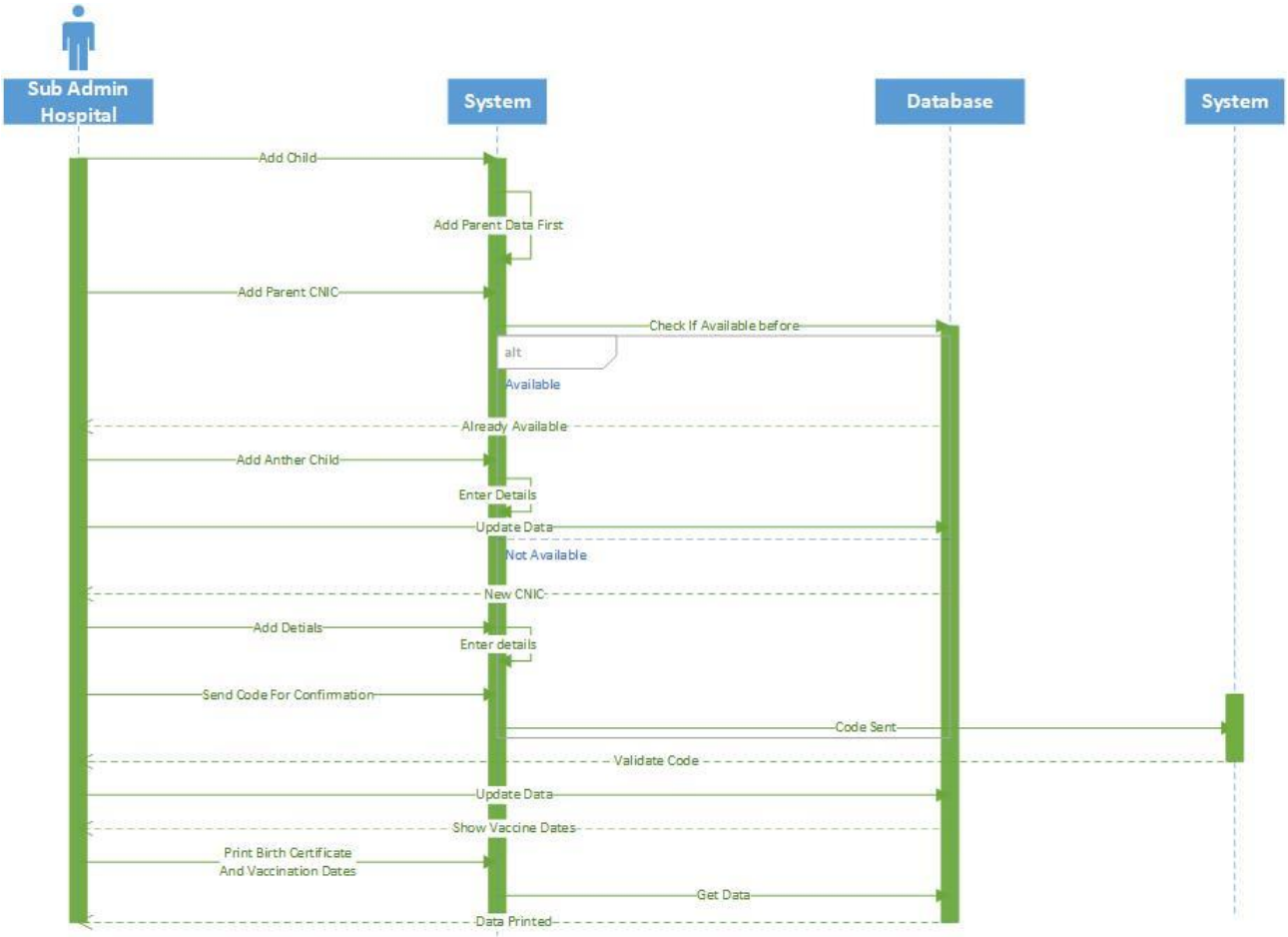

Figure 12: Sequence Diagram For Add Child Data

4.2.2 Sequence Diagram For Campaign Creation

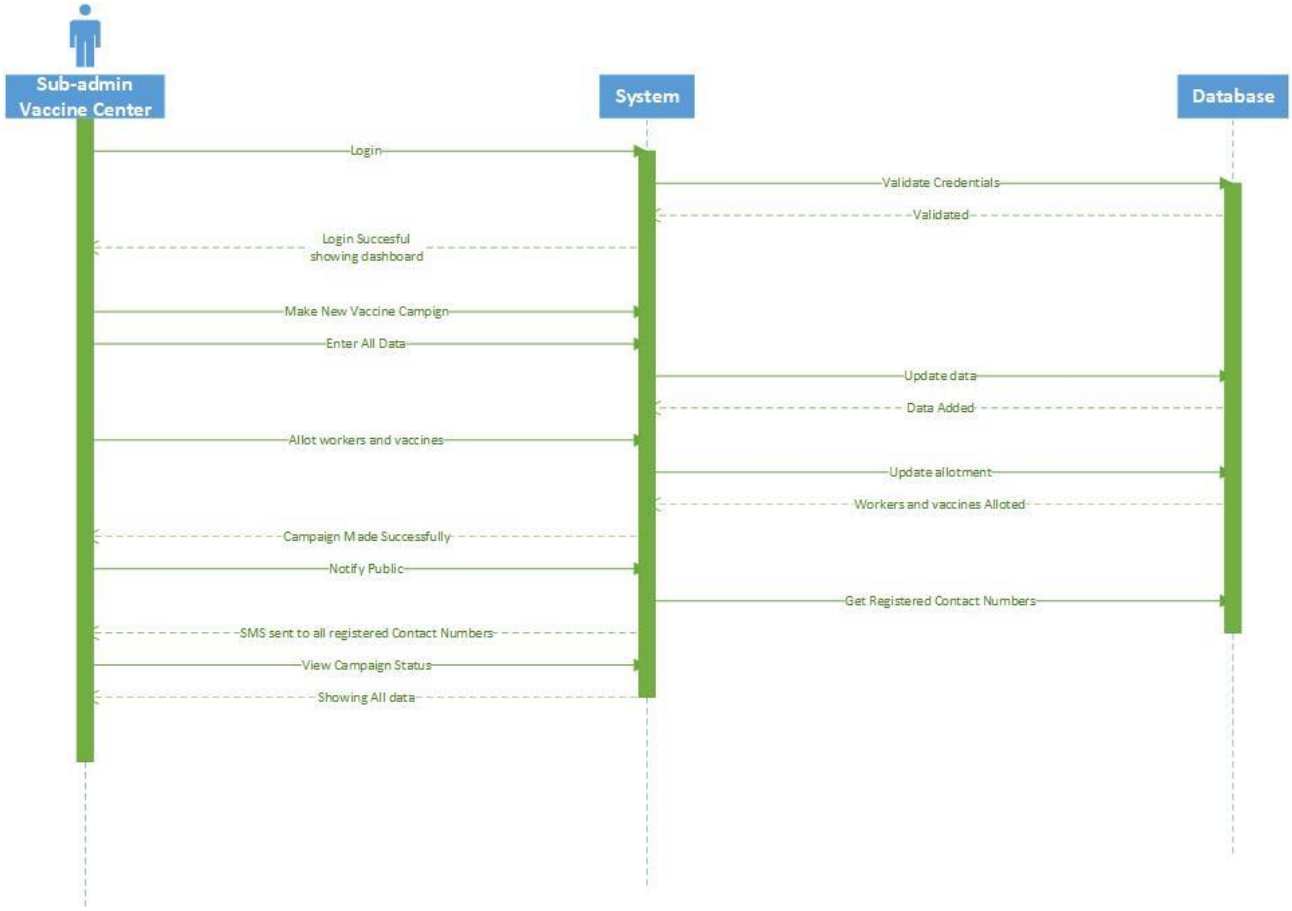

Figure 13: Sequence Diagram For Campaign Creation

4.2.3 Sequence Diagram For Child Growth Prediction

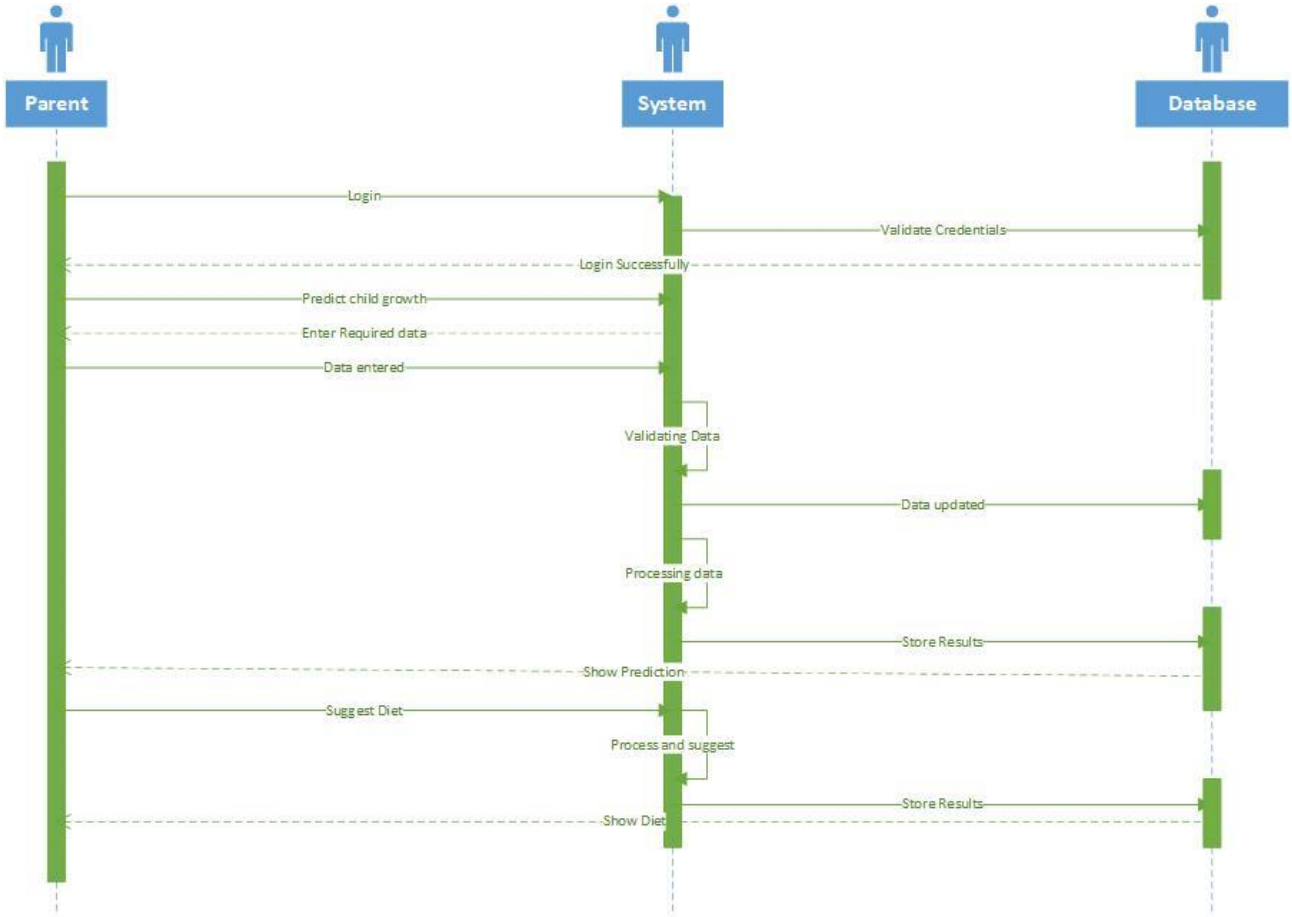

Figure 14: Sequence Diagram For Child Growth Prediction

4.2.4 Sequence Diagram For Child Vaccine Update

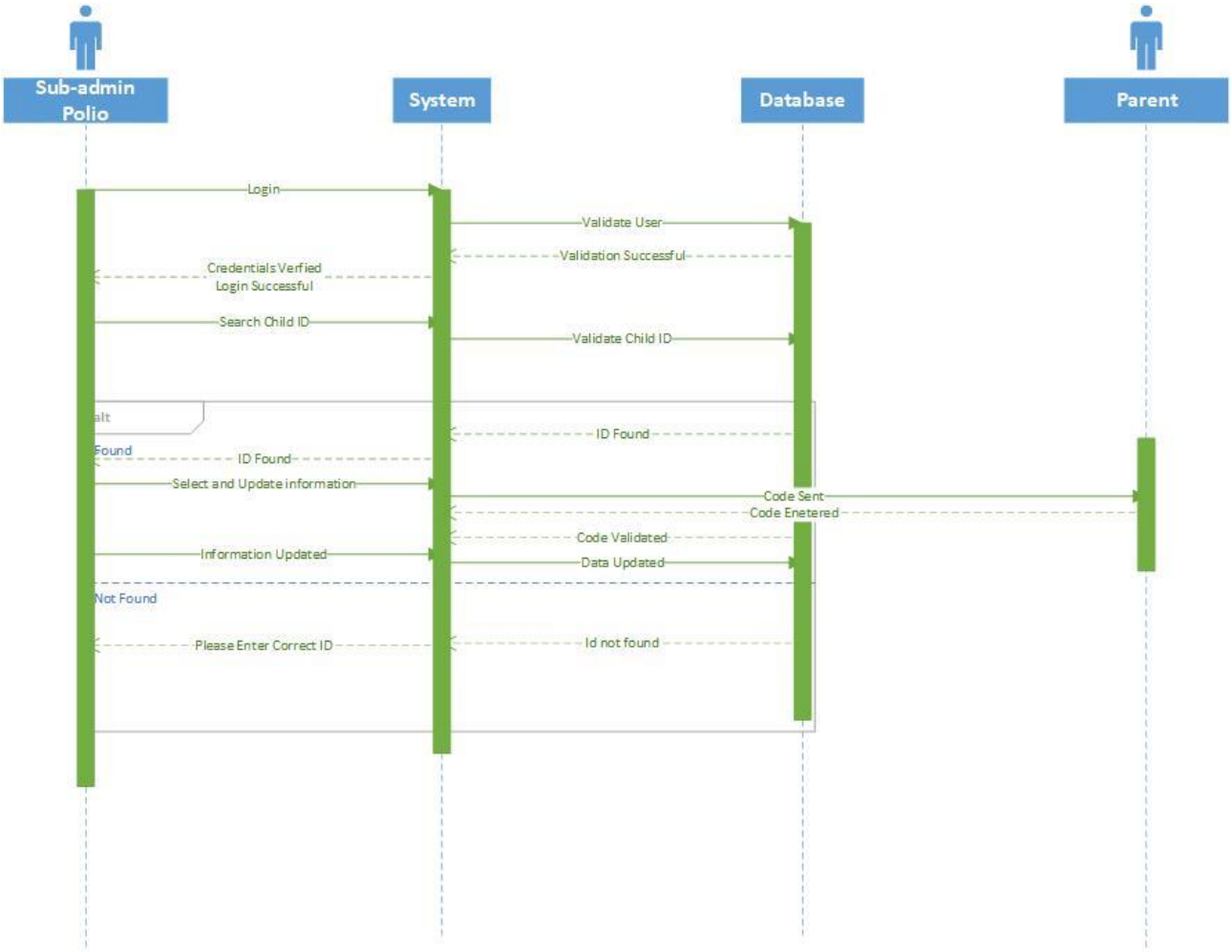

Figure 15: Sequence Diagram For Child Vaccine Update

4.2.5 Sequence Diagram For Predict Vaccine Stock

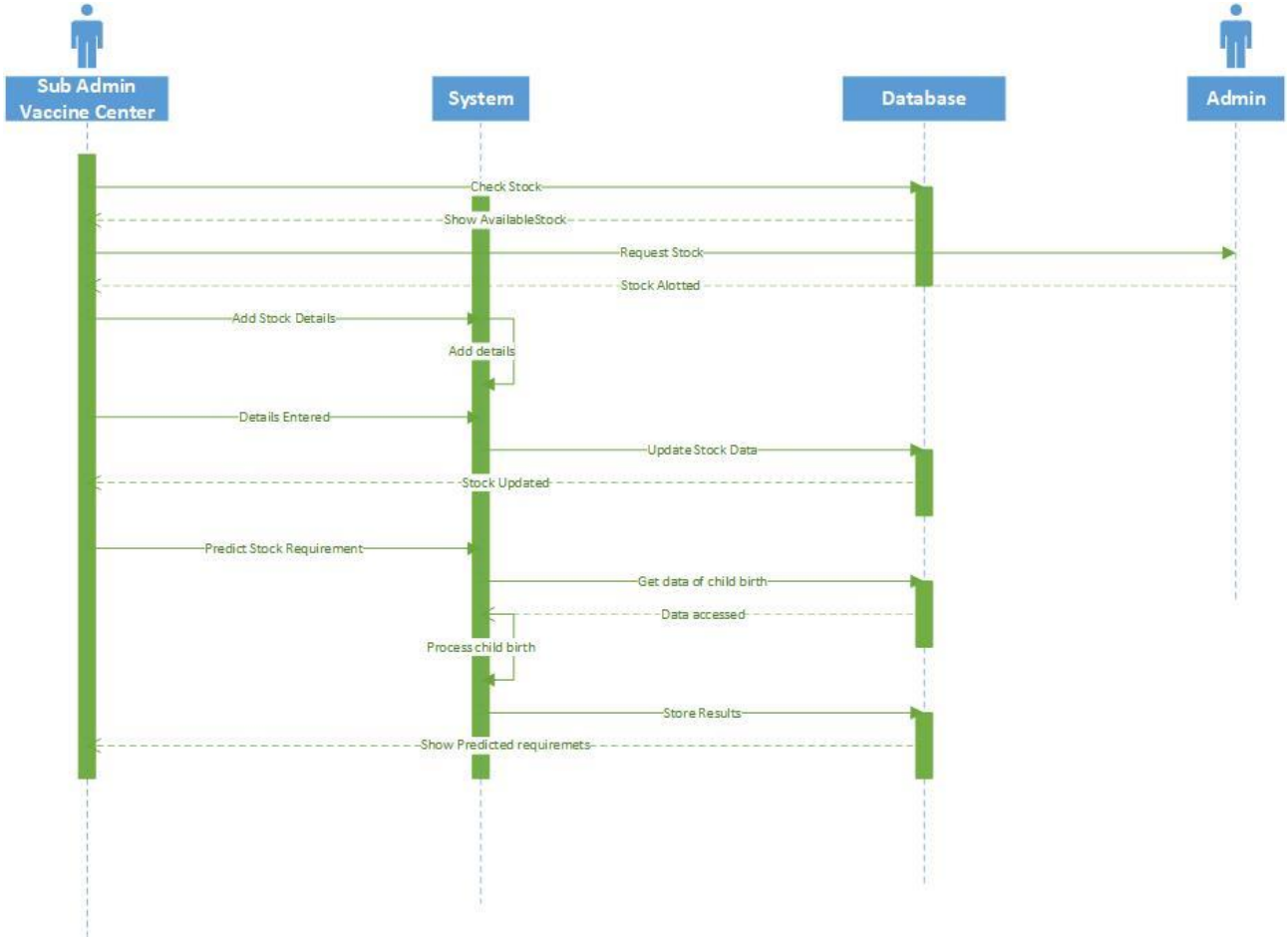

Figure 16: Sequence Diagram For Predict Vaccine Stock

4.2.6 Sequence Diagram For Polio Vaccine update and Symptoms Check

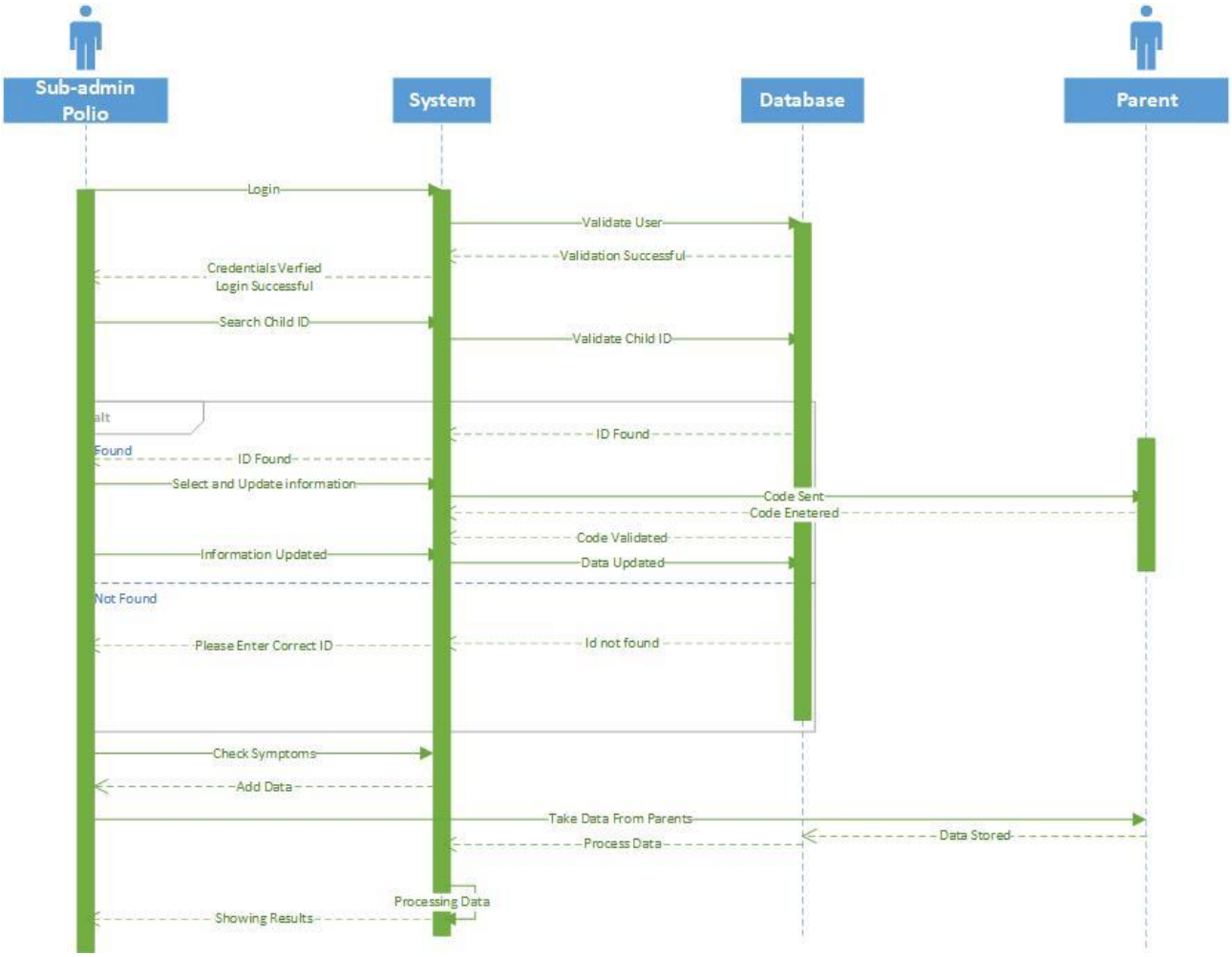

Figure 17: Sequence Diagram For Polio Vaccine Update

5. Data design

5.1 ERD

ERD for Full System

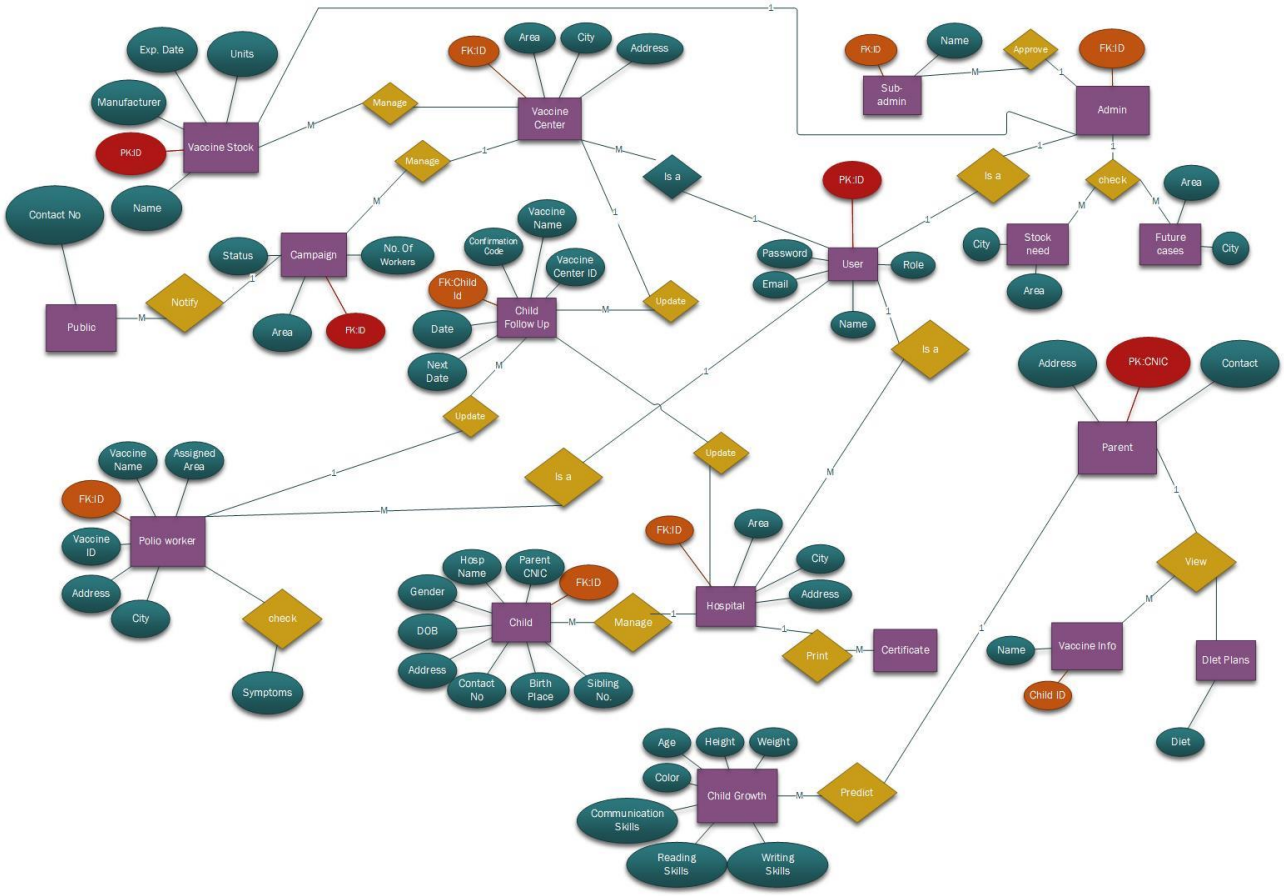

Figure 18: ERD

5.2 Data dictionary

5.2.1 Parent Schema

```
const parentSchema = new Schema(
   {
      parentID: { type: String, default: "vaccine center", required: true },
      role: {
         type: String,
         default: "parent",
         required: true,
         enum: ["vaccine center", "hospital", "parent", "polio worker"],
      },
      name: { type: String, default: "", required: true },
```

```
      email: { type: String, default: "", required: true },
      password: { type: String, default: "", required: true },
      contactNo: { type: String, required: true, default: "" },
      address: { type: String, required: true, default: "" },
      cnic: { type: String, required: true, default: "" },
   },
   {
      timestamps: true,
   }
);
```

5.2.2 Vaccine Schema

```
const vaccineSchema = new Schema(
   {
      vaccineID: { type: String, default: "", required: true },
      name: { type: String, default: "", required: true, enum: ["active",
"inactive"] },
      manufacturer: { type: String, default: "", required: true },
      quantity: { type: Number, default: 50, required: true },
      expiryDate: { type: Date, default: Date.now, required: true },
   },
   {
      timestamps: true,
   }
);
```

5.2.3 User Schema

```
const userSchema = new Schema(
   {
      userID: { type: String, default: "", required: true },
      role: {
         type: String,
         default: "",
         required: true,
         enum: ["vaccine center", "hospital", "parent", "polio worker"],
      },
      name: { type: String, default: "", required: true },
      email: { type: Number, default: 5, required: true },
      password: { type: Date, default: Date.now },
   },
   {
      timestamps: true,
   }
);
```

5.2.4 Polio Worker Schema

```
const polioWorkerSchema = new Schema(
   {
      workerID: { type: String, default: "vaccine center", required: true },
```

```
      role: {
         type: String,
         default: "polio worker",
         required: true,
         enum: ["vaccine center", "hospital", "parent", "polio worker"],
      },
      name: { type: String, default: "", required: true },
      email: { type: String, default: "", required: true },
      password: { type: String, default: "", required: true },
      contactNo: { type: String, required: true, default: "" },
      city: { type: String, required: true, default: "" },
      assignedArea: { type: String, required: true, default: "" },
      vaccineCenter: { type: String, required: true, default: "" },
   },
   {
      timestamps: true,
   }
);
```

5.2.5 Vaccine Schema

```
const vaccineSchema = new Schema(
   {
      vaccineID: { type: String, default: "", required: true },
      name: { type: String, default: "", required: true, enum: ["active",
"inactive"] },
      manufacturer: { type: String, default: "", required: true },
      quantity: { type: Number, default: 50, required: true },
      expiryDate: { type: Date, default: Date.now, required: true },
   },
   {
      timestamps: true,
   }
);
```

5.2.6 Hospital Schema

```
const hospitalSchema = new Schema(
   {
      hospitalID: { type: String, default: "hospital", required: true },
      role: {
         type: String,
         default: "",
         required: true,
         enum: ["vaccine center", "hospital", "parent", "polio worker"],
      },
      name: { type: String, default: "", required: true },
      email: { type: String, default: "", required: true },
      password: { type: String, default: "", required: true },
      area: { type: String, default: "", required: true },
      city: { type: String, default: "", required: true },
      address: { type: String, default: "", required: true },
   },
   {
```

```
      timestamps: true,
   }
);
```

5.2.7 Child Schema

```
const childSchema = new Schema(
   {
      childID: { type: String, required: true, default: "" },
      parentName: { type: String, required: true, default: "" },
      parentCNIC: { type: String, required: true, default: "" },
      contactNo: { type: String, required: true, default: "" },
      address: { type: String, required: true, default: "" },
      dateOfBirth: { type: Date, required: true, default: "" },
      gender: { type: String, required: true, default: "" },
      birthPlace: { type: String, required: true, default: "" },
      siblingNo: { type: Number, required: true, default: "" },
      hospitalName: { type: String, required: true, default: "" },
      vaccinationInfo: { type: String, default: "" },
   },
   {
      timestamps: true,
   }
);
```

5.2.8 Campaign Schema

```
const campaignSchema = new Schema(
   {
      campaignID: { type: String, default: "", required: true },
      status: { type: String, default: "active", required: true, enum:
["active", "inactive"] },
      area: { type: String, default: "", required: true },
      noOfWorkers: { type: Number, default: 5, required: true },
      startDate: { type: Date, default: Date.now },
      endDate: { type: Date, default: Date.now },
   },
   {
      timestamps: true,
   }
);
```

6. Algorithm & Implementation

6.1 Implementation

6.1.1 Server

```
const express = require("express"),
   cors = require("cors"),
```

```
   mongoose = require("mongoose");

require("dotenv").config();

const app = express();
const PORT = process.env.PORT || 5000;

app.use(cors());
app.use(express.json());

// db connection
const ATLAS_URI = process.env.ATLAS_URI;
mongoose.connect(ATLAS_URI, {
   useNewUrlParser: true,
   useCreateIndex: true,
   useUnifiedTopology: true,
});

const connection = mongoose.connection;
connection.once("open", () => {
   console.log(`Mongodb connected successfully`);
});

// routers
const childRouter = require("./routes/children");
const vaccineRouter = require("./routes/vaccines");
const campaignRouter = require("./routes/campaigns");

app.use("/children", childRouter);
app.use("/vaccines", vaccineRouter);
app.use("/campaigns", campaignRouter);

// start server
app.listen(PORT, () => {
   console.log(`Server is running at ${PORT}`);
});
```

6.1.2 Campaign Routing

```
const router = require("express").Router();
let Campaign = require("../models/Campaign.model");

// get Campaigns
router.route("/").get((req, res) => {
   Campaign.find()
      .then((campaign) => res.json(campaign))
      .catch((err) => res.status(400).json({ error: err }));
});

// get single Campaign
router.route("/:id").get((req, res) => {
   Campaign.findById(req.params.id)
      .then((campaign) => res.json(campaign))
      .catch((err) => res.status(400).json({ error: err }));
});

// delete Campaign
```

```
router.route("/:id").delete((req, res) => {
   Campaign.findByIdAndDelete(req.params.id)
      .then((campaign) => res.json("Campaign deleted "))
      .catch((err) => res.status(400).json({ error: err }));
});

// add Campaign
router.route("/add").post((req, res) => {
   const campaignID = req.body.campaignID;
   const status = req.body.status;
   const area = req.body.area;
   const noOfWorkers = Number(req.body.noOfWorkers);
   const startDate = Date.parse(req.body.startDate);
   const endDate = Date.parse(req.body.endDate);

   const newCampaign = new Campaign({
      campaignID,
      status,
      area,
      noOfWorkers,
      startDate,
      endDate,
   });

   newCampaign
      .save()
      .then(() => res.json("Campaign added!"))
      .catch((err) => res.status(400).json({ error: err }));
});

// update Campaign
router.route("/update/:id").post((req, res) => {
   Campaign.findById(req.params.id)
      .then((campaign) => {
         campaign.campaignID = req.body.campaignID;
         campaign.status = req.body.status;
         campaign.area = req.body.area;
         campaign.noOfWorkers = Number(req.body.noOfWorkers);
         campaign.startDate = Date.parse(req.body.startDate);
         campaign.endDate = Date.parse(req.body.endDate);

         campaign
            .save()
            .then(() => res.json("Campaign updated!"))
            .catch((err) => res.status(400).json({ error: err }));
      })
      .catch((err) => res.status(400).json({ error: err }));
});

module.exports = router;
```

6.1.3 Child Routing

```
const router = require("express").Router();
let Child = require("../models/child.model");

// get children
```

```
router.route("/").get((req, res) => {
   Child.find()
      .then((children) => res.json(children))
      .catch((err) => res.status(400).json({ error: err }));
});

// get single child
router.route("/:id").get((req, res) => {
   Child.findById(req.params.id)
      .then((children) => res.json(children))
      .catch((err) => res.status(400).json({ error: err }));
});

// delete child
router.route("/:id").delete((req, res) => {
   Child.findByIdAndDelete(req.params.id)
      .then((children) => res.json("Child deleted "))
      .catch((err) => res.status(400).json({ error: err }));
});

// add child
router.route("/add").post((req, res) => {
   const childID = req.body.childID;
   const parentName = req.body.parentName;
   const parentCNIC = req.body.parentCNIC;
   const contactNo = req.body.contactNo;
   const address = req.body.address;
   const dateOfBirth = Date.parse(req.body.dateOfBirth);
   const gender = req.body.gender;
   const birthPlace = req.body.birthPlace;
   const siblingNo = Number(req.body.siblingNo);
   const hospitalName = req.body.hospitalName;
   const vaccinationInfo = req.body.vaccinationInfo;

   const newChild = new Child({
      childID,
      parentName,
      parentCNIC,
      contactNo,
      address,
      dateOfBirth,
      gender,
      birthPlace,
      siblingNo,
      hospitalName,
      vaccinationInfo,
   });

   newChild
      .save()
      .then(() => res.json("Child added!"))
      .catch((err) => res.status(400).json({ error: err }));
});

// update child
router.route("/update/:id").post((req, res) => {
   Child.findById(req.params.id)
```

```
      .then((child) => {
         child.childID = req.params.childID;
         child.parentName = req.body.parentName;
         child.parentCNIC = req.body.parentCNIC;
         child.childID = req.body.childID;
         child.contactNo = req.body.contactNo;
         child.address = req.body.address;
         child.dateOfBirth = Date.parse(req.body.dateOfBirth);
         child.gender = req.body.gender;
         child.birthPlace = req.body.birthPlace;
         child.siblingNo = Number(req.body.siblingNo);
         child.hospitalName = req.body.hospitalName;
         child.vaccinationInfo = req.body.vaccinationInfo;

         child
            .save()
            .then(() => res.json("Child updated!"))
            .catch((err) => res.status(400).json({ error: err }));
      })
      .catch((err) => res.status(400).json({ error: err }));
});

module.exports = router;
```

6.1.4 Vaccine Routing

```
const router = require("express").Router();
let Vaccine = require("../models/Vaccine.model");

// get Vaccines
router.route("/").get((req, res) => {
   Vaccine.find()
      .then((vaccine) => res.json(vaccine))
      .catch((err) => res.status(400).json({ error: err }));
});

// get single Vaccine
router.route("/:id").get((req, res) => {
   Vaccine.findById(req.params.id)
      .then((vaccine) => res.json(vaccine))
      .catch((err) => res.status(400).json({ error: err }));
});

// delete Vaccine
router.route("/:id").delete((req, res) => {
   Vaccine.findByIdAndDelete(req.params.id)
      .then((vaccine) => res.json("Vaccine deleted "))
      .catch((err) => res.status(400).json({ error: err }));
});

// add Vaccine
router.route("/add").post((req, res) => {
   const vaccineID = req.body.vaccineID;
   const name = req.body.name;
   const manufacturer = req.body.manufacturer;
```

```
    const quantity = Number(req.body.quantity);
    const expiryDate = Date.parse(req.body.expiryDate);

    const newVaccine = new Vaccine({
        vaccineID,
        name,
        manufacturer,
        quantity,
        expiryDate,
    });

    newVaccine
        .save()
        .then(() => res.json("Vaccine added!"))
        .catch((err) => res.status(400).json({ error: err }));
});

// update Vaccine
router.route("/update/:id").post((req, res) => {
    Vaccine.findById(req.params.id)
        .then((vaccine) => {
            vaccine.vaccineID = req.body.vaccineID;
            vaccine.name = req.body.name;
            vaccine.manufacturer = req.body.manufacturer;
            vaccine.quantity = Number(req.body.quantity);
            vaccine.expiryDate = Date.parse(req.body.expiryDate);

            vaccine
                .save()
                .then(() => res.json("Vaccine updated!"))
                .catch((err) => res.status(400).json({ error: err }));
        })
        .catch((err) => res.status(400).json({ error: err }));
});

module.exports = router;
```

7. Software requirements traceability matrix

Table 1 Requirements Traceability Matrix

Req. Number	Ref. Item	Design Component	Component Items
FR01	Class Diagram	Admin	InputInformation
FR02	Class Diagram	Admin	Prediction
FR03	Class Diagram	Admin	Predict Vaccine Stock
FR04	Class Diagram	Admin	Predict future cases
FR05	Class Diagram	Admin	Predict future cases by city, area

FR06	Class Diagram	Admin	Predict future stock needs by city, area
FR07	Class Diagram	Admin	View Stats
FR08	Class Diagram	Admin	View Stats by city/area
FR09	Class Diagram	Admin	Search Stats
FR10	Class Diagram	Admin	Search Stock by city/area
FR11	Class Diagram	Admin	Search Cases by city/area
FR12	Class Diagram	Admin	Search child info
FR13	Class Diagram	Admin	Manage sub admins
FR14	Class Diagram	Admin	Add sub admin
FR15	Class Diagram	Admin	Remove sub admin
FR16	Class Diagram	Admin	View sub admin progress
FR17	Class Diagram	Parent	Fill information
FR18	Class Diagram	Parent	Input information
FR19	Class Diagram	Parent	Click on View child details
FR20	Class Diagram	Parent	Enter child ID
FR21	Class Diagram	Parent	Receive notifications
FR22	Class Diagram	Parent	Confirm vaccinations
FR23	Class Diagram	Parent	Check child growth
FR24	Class Diagram	Parent	Fill information
FR25	Class Diagram	Parent	Submit Information
FR26	Class Diagram	Parent	View suggested diet
FR27	Class Diagram	Parent	Update settings
FR28	Class Diagram	Parent	Update password
FR29	Class Diagram	Parent	Update child record
FR30	Class Diagram	Parent	Enter Symptoms
FR31	Class Diagram	Parent	Check Polio Symptoms
FR32	Class Diagram	Parent	See Results
FR33	Class Diagram	Polio Worker	Send notifications
FR34	Class Diagram	Polio Worker	Receive confirmation
FR35	Class Diagram	Polio Worker	Update status
FR36	Class Diagram	Polio Worker	Enter Symptoms
FR37	Class Diagram	Polio Worker	Check Polio Symptoms
FR38	Class Diagram	Polio Worker	See Results

FR39	Class Diagram	Polio Worker	Report Cases
FR40	Class Diagram	Hospital, Vaccination center	Sign up from web app.
FR41	Class Diagram	Hospital, Vaccination center	Enter Information
FR42	Class Diagram	Hospital, Vaccination center	Login
FR43	Class Diagram	Hospital	add child record
FR44	Class Diagram	Hospital	Enter child details
FR45	Class Diagram	Hospital	Submit child record
FR46	Class Diagram	Hospital, Vaccination center	Sent Confirmation
FR47	Class Diagram	Hospital, Vaccination center	Receive Confirmation
FR48	Class Diagram	Hospital, Vaccination center	update child record
FR49	Class Diagram	Hospital, Vaccination center	Search ID
FR50	Class Diagram	Hospital, Vaccination center	Submit updated child record
FR51	Class Diagram	Hospital	print birth certificate
FR52	Class Diagram	Hospital	Adjust print options
FR53	Class Diagram	Hospital	print vaccination dates
FR54	Class Diagram	Hospital, Vaccination center	Search child information
FR55	Class Diagram	Hospital, Vaccination center	Check vaccine requirements
FR56	Class Diagram	Hospital, Vaccination center	Update child vaccination record
FR57	Class Diagram	Hospital, Vaccination center	Report authorities
FR58	Class Diagram	Hospital, Vaccination center	Manage vaccine stock
FR59	Class Diagram	Hospital, Vaccination center	Add stock
FR60	Class Diagram	Hospital, Vaccination center	update stock
FR61	Class Diagram	Hospital, Vaccination center	View stock

FR62	Class Diagram	Hospital, Vaccination center	Request for stock
FR63	Class Diagram	Vaccination center	Manage campaigns
FR64	Class Diagram	Vaccination center	Make new campaigns
FR65	Class Diagram	Vaccination center	View campaign status
FR66	Class Diagram	Vaccination center	Allot workers for campaign
FR67	Class Diagram	Vaccination center	Notify public

8. Human interface design

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.

8.1 Screen images

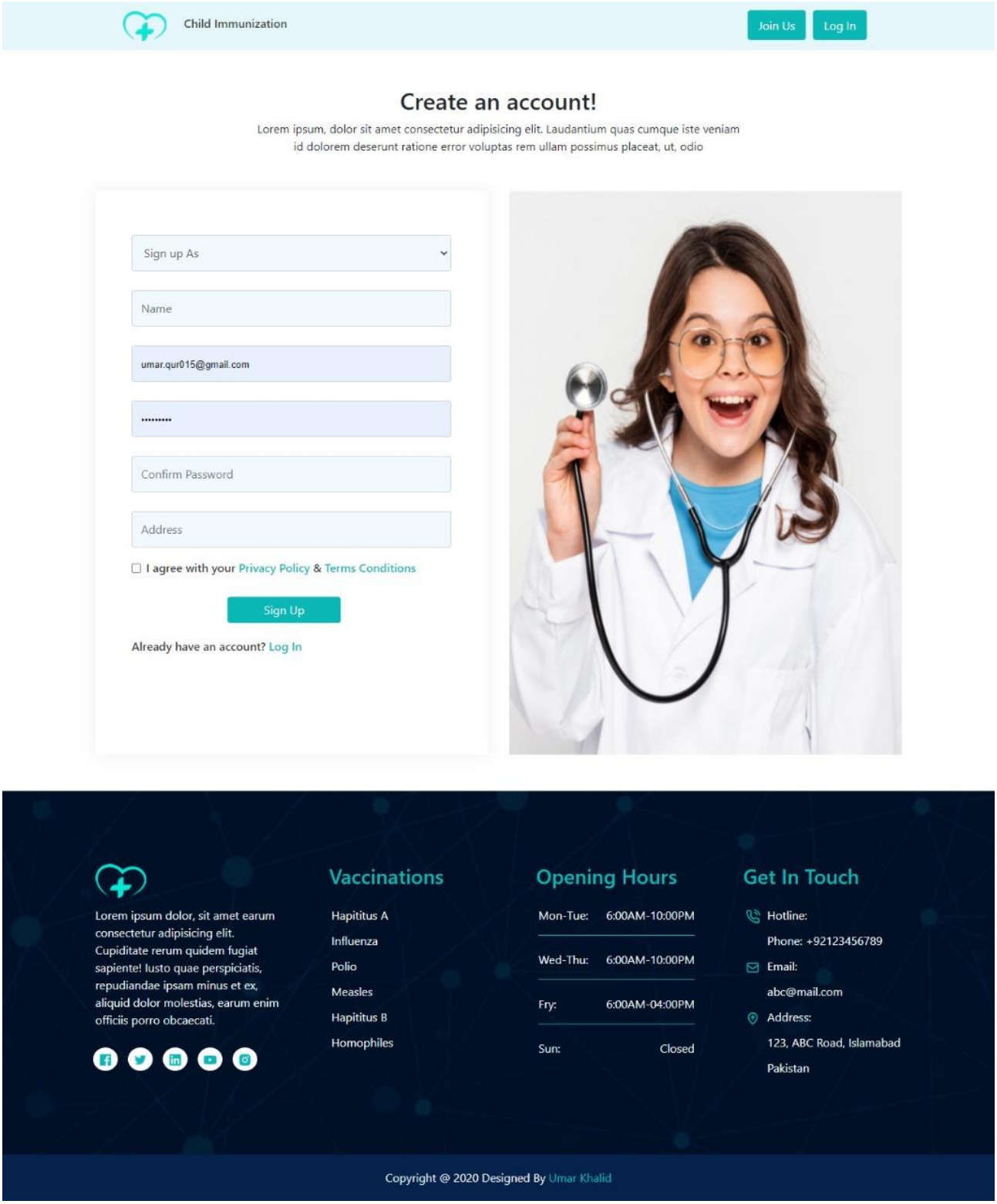

Figure 19: Screen 1

Figure 20: Screen 2

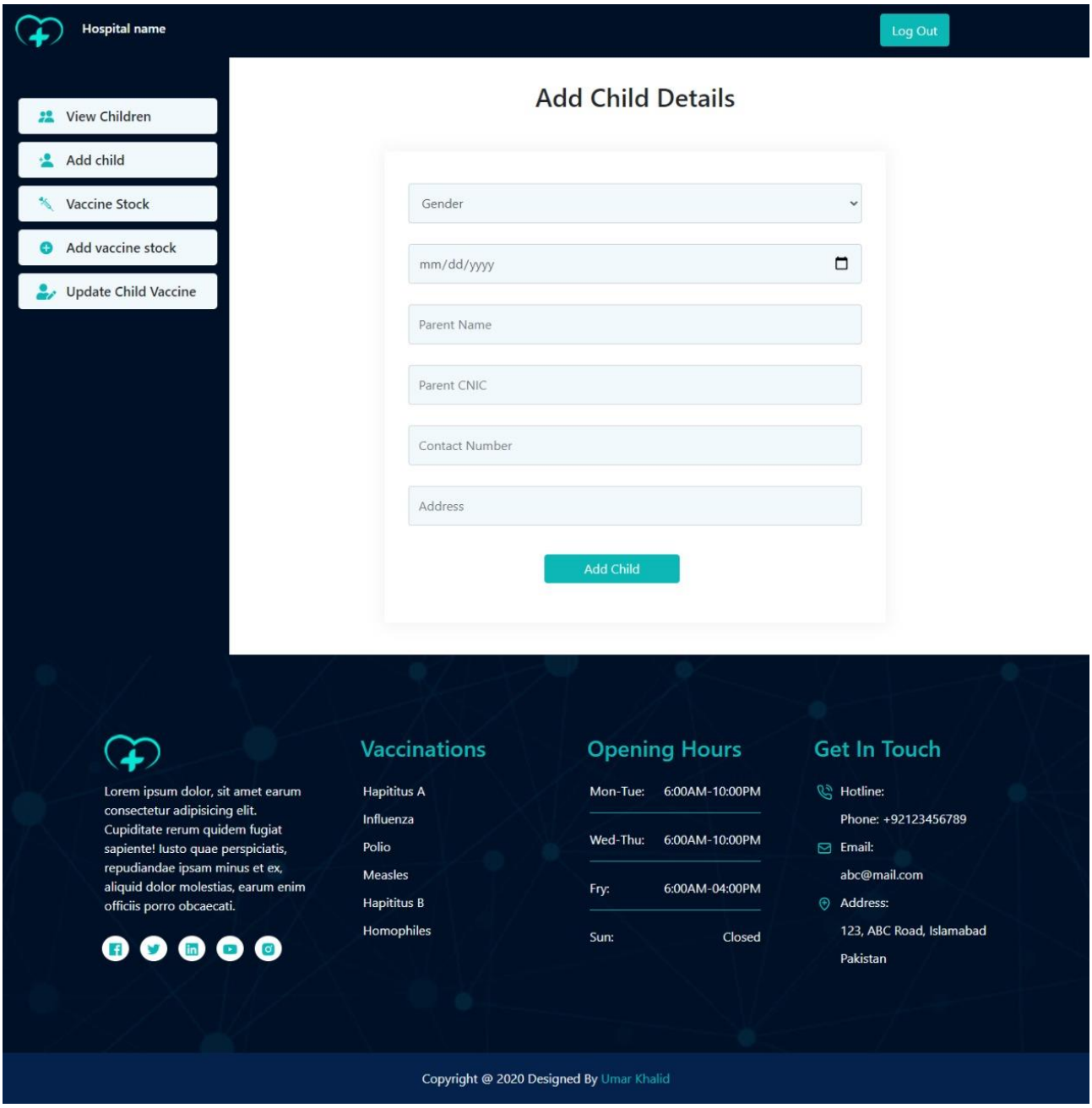

Figure 21: Screen 3

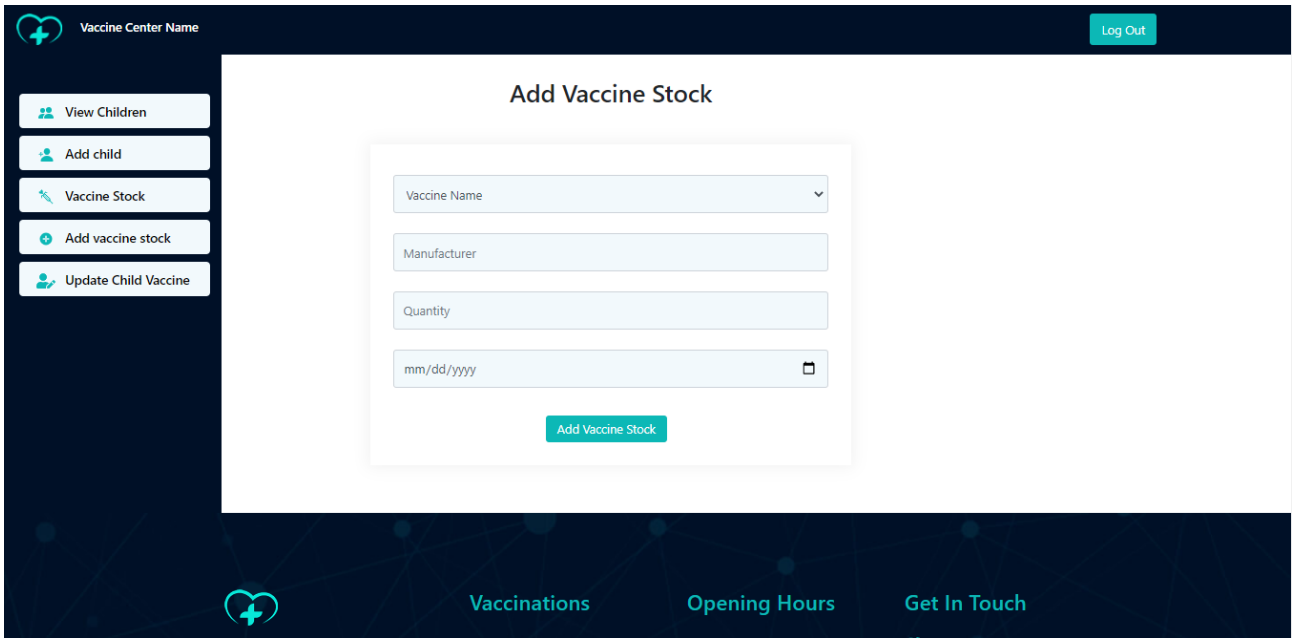

Figure 22: Screen 4

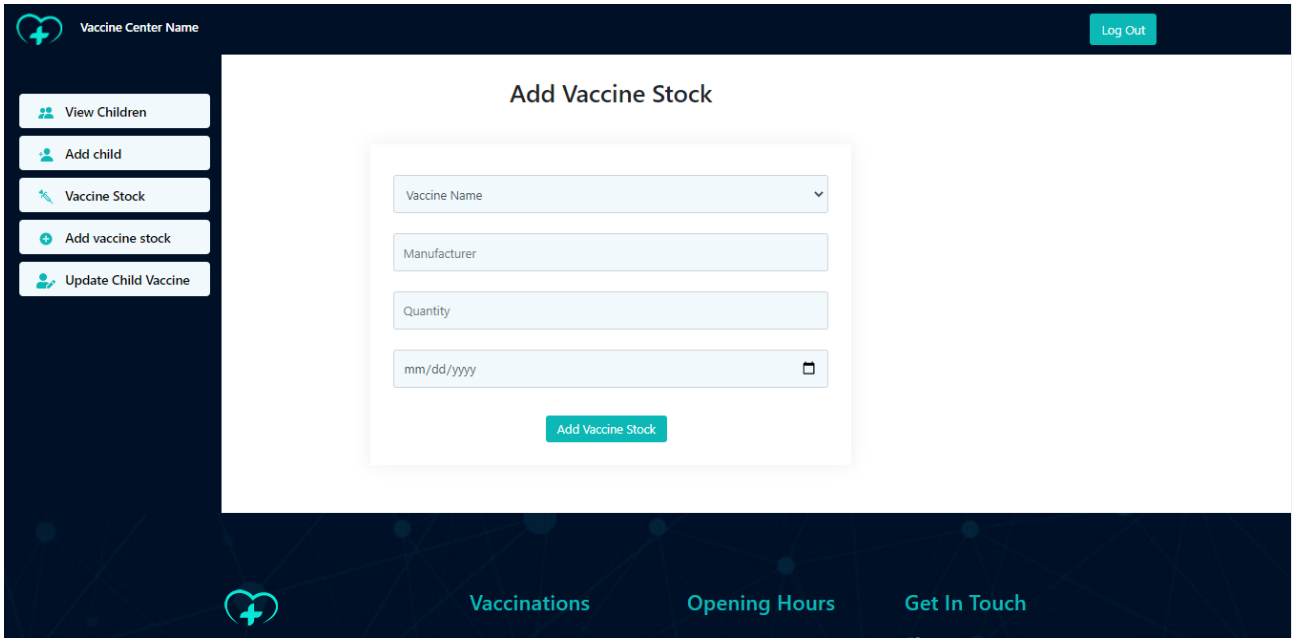

Figure 23: Screen 5