A REPORT ON WEB DEVELOPMENT

Submitted
In partial fulfillment
For the certificate of internship on fullstack development

Submitted by:
B.Shashank
221710314004

Submitted to:
Fullstack development team
Dhyanahitha institution

Department of computer science and engineering

CANDIDATE DECLARATION

I here by declare that the project entitled “MEDICINO(Hospital management) submitted by me to Dhyanahitha institution in partial fulfillment of the requirements for the certificate of internship on fullstack development is a record of bonafide project work carried out by me under guidance of my mentor.I further declare that the work reported in this project has not been submitted and will not be submitted,either in part or in full,for the award or certificate of any other degree.

Signature of candidate:

B.Shashank
221710314004

ACKNOWLEDGEMENT

I would like to express my special profound gratitude to my trainer as well as our mentor who gave me the golden opportunity to do this wonderful project on the topic (Hospital management), which also helped me in doing a lot of Research on the bridge of practical and theoretical and I came to know about so many new things regarding different technologies I am sincerely thankful to them.

Secondly I would also like to thank my team who helped me a lot in finalizing this project within the limited time frame.

ABSTRACT

Web development is the work involved in developing a website for the Internet (World Wide Web) or an intranet (a private network).Web development can range from developing a simple single static page of plain text to complex web-based internet applications (web apps), electronic businesses, and social network services. A more comprehensive list of tasks to which web development commonly refers, may include web engineering, web design, web content development, client liaison, client-side/server-side scripting, web server and network security configuration, and e-commerce development.

Among web professionals, "web development" usually refers to the main non-design aspects of building websites: writing markup and coding. Web development may use content management systems (CMS) to make content changes easier and available with basic technical skills.

For larger organizations and businesses, web development teams can consist of hundreds of people (web developers) and follow standard methods like Agile methodologies while developing websites. Smaller organizations may only require a single permanent or contracting developer, or secondary assignment to related job positions such as a graphic designer or information systems technician. Web development may be a collaborative effort between departments rather than the domain of a designated department. There are three kinds of web developer specialization: front-end developer, back-end developer, and full-stack developer. Front-end developers are responsible for behavior and visuals that run in the user browser, while back-end developers deal with the servers.

Table of content

List of images

List of tables

CHAPTER-1
INTRODUCTION

Full stack development refers to the development of both front end(client side) and back end(server side) portions of web application.

Full stack web developers have the ability to design complete web applications and websites. They work on the frontend, backend, database and debugging of web applications or websites.

Fig.1.1

So full stack development is classified into two phases:

1.FRONTEND

2.BACKEND

CHAPTER-2
Back end development

Back-end Development refers to server-side development. It is the term used for the behind-the-scenes activities that happen when performing any action on a website. It can be logging in to your account or purchasing a watch from an online store.

Backend developer focuses on databases, scripting, and the architecture of websites. Code written by back-end developers helps to communicate the database information to the browser.

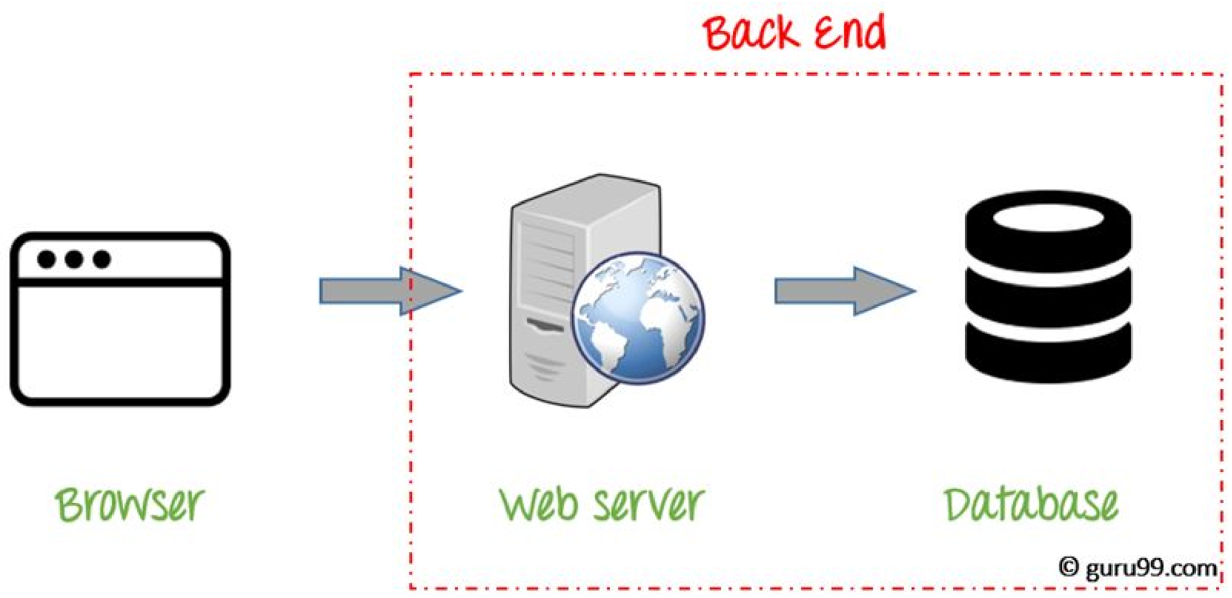

Fig.2.1

Backend languages:

1.SQL
2.Query
3.Database
4.Node js

2.1 SQL

SQL is a database computer language designed for the retrieval and management of data in a relational database. SQL stands for Structured Query Language. This tutorial will give you a quick start to SQL. It covers most of the topics required for a basic understanding of SQL and to get a feel of how it works.SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.SQL is the standard language for the Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Also, they are using different dialects, such as −

1.MS SQL Server using T-SQL,

2.Oracle using PL/SQL,

3.MS Access version of SQL is called JET SQL (native format) etc.

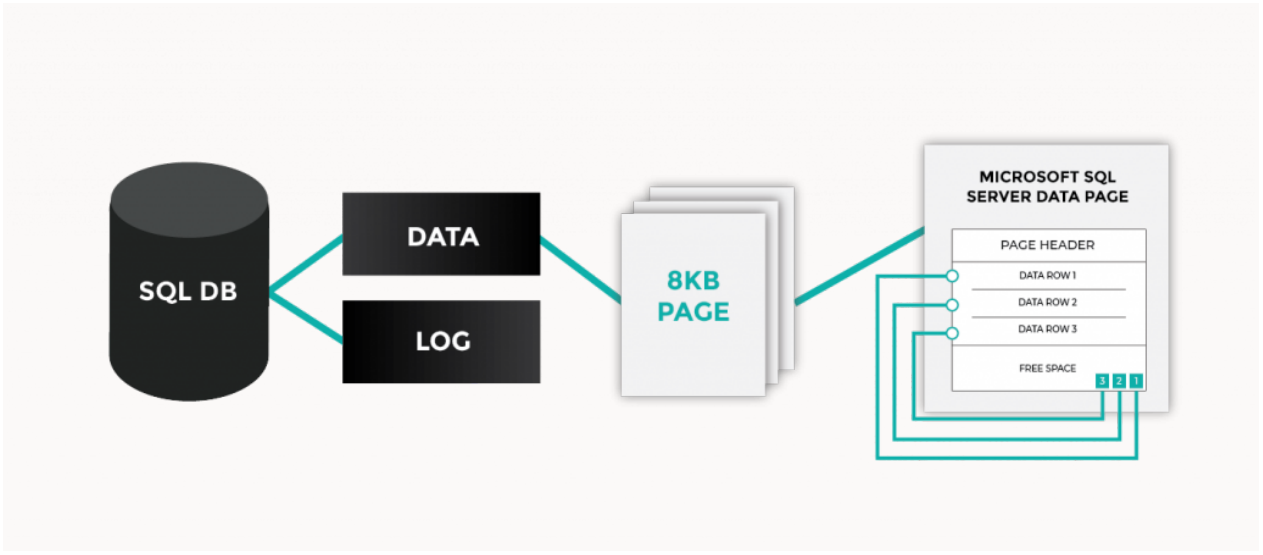

Fig.2.1.1

2.2.JQUERY

A query is a question, often expressed in a formal way. A database query can be either a select query or an action query. A select query is a data retrieval query, while an action query asks for additional operations on the data, such as insertion, updating or deletion.

Query languages are used to make queries in a database, and Microsoft Structured Query Language (SQL) is the standard. Under the SQL query umbrella, there are several extensions of the language, including MySQL, Oracle SQL and NuoDB.

Query languages for other types of databases, such as NoSQL databases and graph databases, include Cassandra Query Language (CQL), Neo4j's Cypher, Data Mining Extensions (DMX) and XQuery.

Queries can accomplish a few different tasks. Primarily, queries are used to find specific data by filtering specific criteria. Queries can also calculate or summarize data, as well as automate data management tasks. Other queries include parameter, totals, crosstab, make table, append, update and delete. For example, a parameter query runs variations of a particular query, which prompts a user to insert a field value, and then it uses that value to create the criteria, while totals queries allow users to group and summarize data.

Different commands in sql query:

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table

- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index
- WHERE-used to select particular column or row

  Still there are more commands in sql query.

2.3 Database

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

Data within the most common types of databases in operation today is typically modeled in rows and columns in a series of tables to make processing and data querying efficient. The data can then be easily accessed, managed, modified, updated, controlled, and organized. Most databases use structured query language (SQL) for writing and querying data.
Types of database:

- Centralised database.
- Distributed database.
- Personal database.
- End-user database.
- Commercial database.
- NoSQL database.
- Operational database.
- Relational database.
- Cloud database.
- Object-oriented database.
- Graph database.

1. Centralised Database:

The information(data) is stored at a centralized location and the users from different locations can access this data. This type of database contains application procedures that help the users to access the data even from a remote location.

Various kinds of authentication procedures are applied for the verification and validation of end users, likewise, a registration number is provided by the application procedures which keeps a track and record of data usage. The local area office handles this thing.

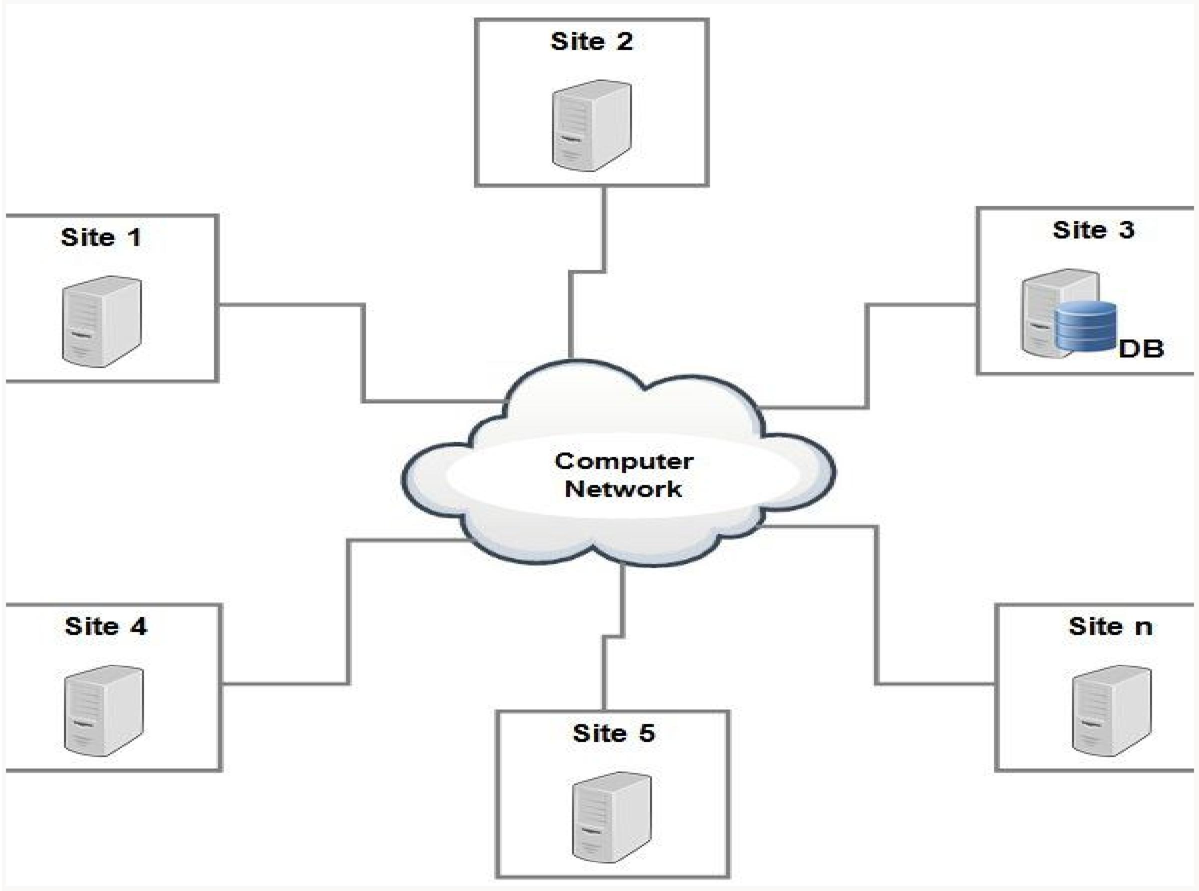

Fig.2.3.1

2.Distributed Database:

Just opposite of the centralized database concept, the distributed database has contributions from the common database as well as the information captured by local computers also. The data is not at one place and is distributed at various sites of an organization. These sites are connected to each other with the help of communication links which helps them to access the distributed data easily.

You can imagine a distributed database as a one in which various portions of a database are stored in multiple different locations(physical) along with the application procedures which are replicated and distributed among various points in a network.

There are two kinds of distributed database, viz. homogenous and heterogeneous. The databases which have same underlying hardware and run over same operating systems and application procedures are known as homogeneous DDB, for eg. All physical locations in a DDB. Whereas, the operating systems, underlying hardware as well as application procedures can be different at various sites of a DDB which is known as heterogeneous DDB.

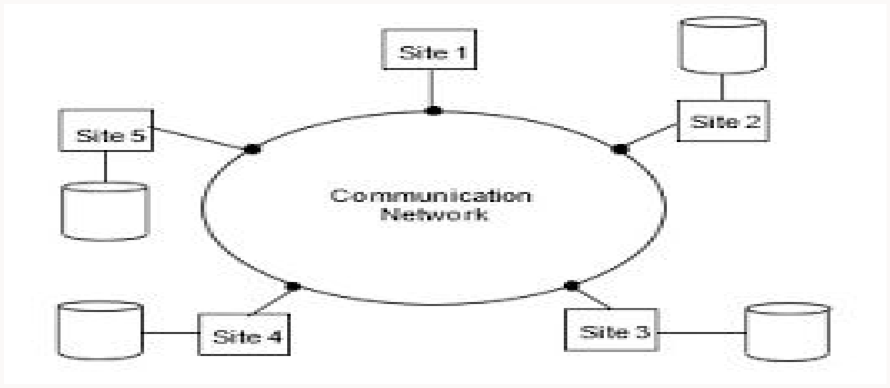

Fig.2.3.2

3.Personal Database:

Data is collected and stored on personal computers which is small and easily manageable. The data is generally used by the same department of an organization and is accessed by a small group of people.

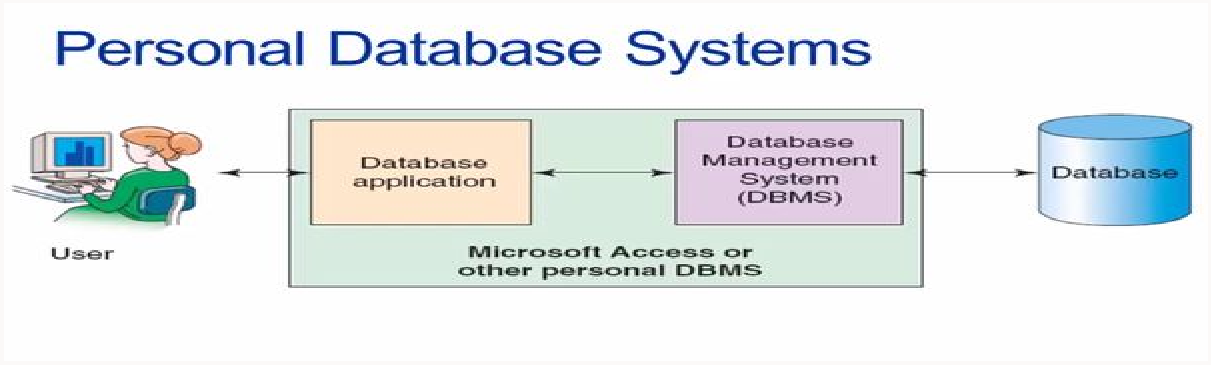
Personal Database Systems
User
Database application
Database Management System (DBMS)
Microsoft Access or other personal DBMS
Database

Fig.2.3.3

4.End User Database:

The end user is usually not concerned about the transaction or operations done at various levels and is only aware of the product which may be a software or an application. Therefore, this is a shared database which is specifically designed for the end user, just like different levels' managers. Summary of the whole information is collected in this database.

5.Commercial Database:

These are the paid versions of the huge databases designed uniquely for the users who want to access the information for help. These databases are subject specific, and one cannot afford to maintain such a huge amount of information. Access to such databases is provided through commercial links.

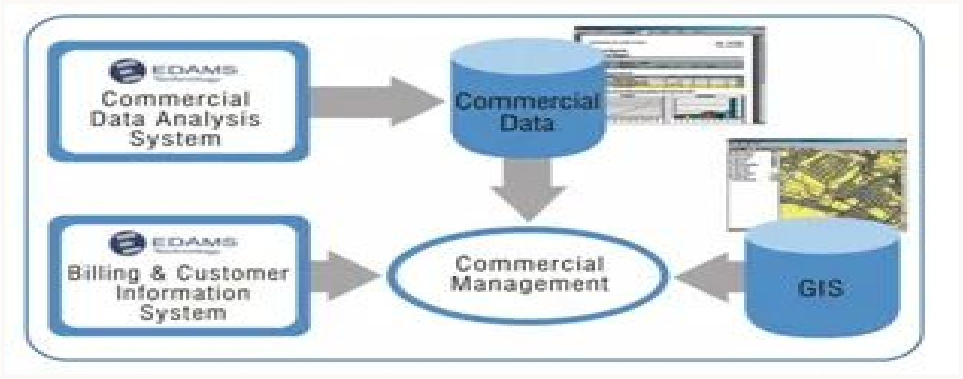
EDAMS
Commercial Data Analysis System
Commercial Data
EDAMS
Billing & Customer Information System
Commercial Management
GIS

Fig.2.3.4

6.NoSQL Database:

These are used for large sets of distributed data. There are some big data performance issues which are effectively handled by relational databases, such kind of issues are easily managed by NoSQL databases. They are very efficient in analyzing large unstructured data that may be stored at multiple virtual servers of the cloud.

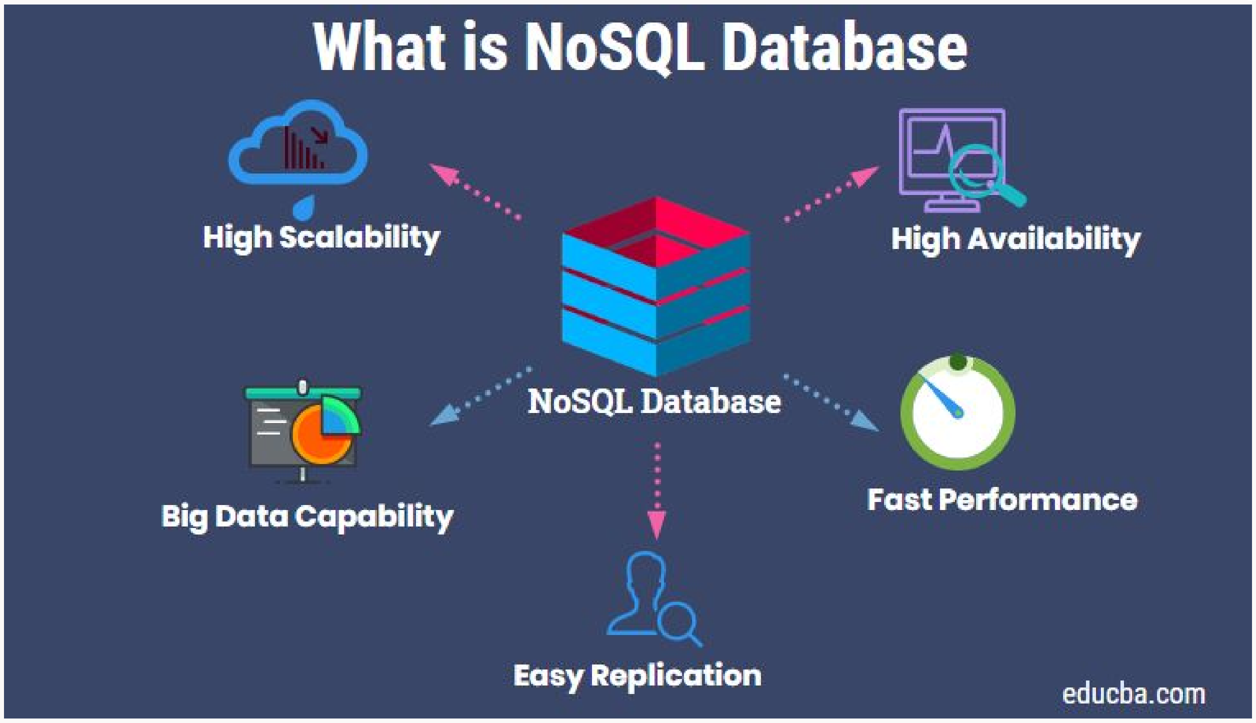

Fig.2.3.5

7.Operational Database:

Information related to operations of an enterprise is stored inside this database. Functional lines like marketing, employee relations, customer service etc. require such kind of databases.

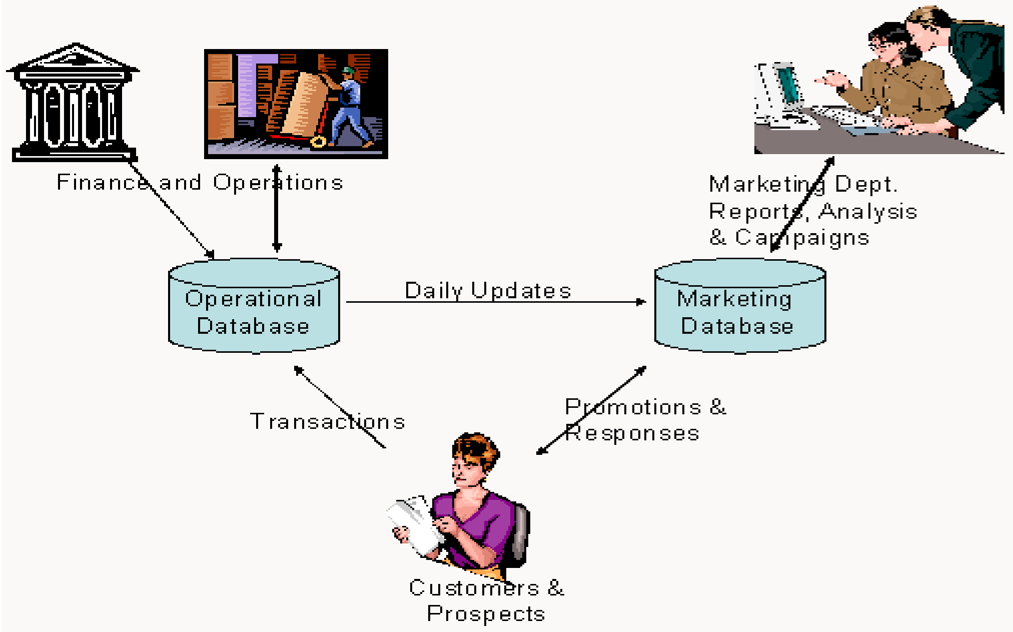

Fig.2.3.6

8.Relational Databases :

These databases are categorized by a set of tables where data gets fit into a pre-defined category. The table consists of rows and columns where the column has an entry for data for a specific category and rows contains instances for that data defined according to the category. The Structured Query Language (SQL) is the standard user and application program interface for a relational database.

There are various simple operations that can be applied over the table which makes these databases easier to extend, join two databases with a common relation and modify all existing applications.

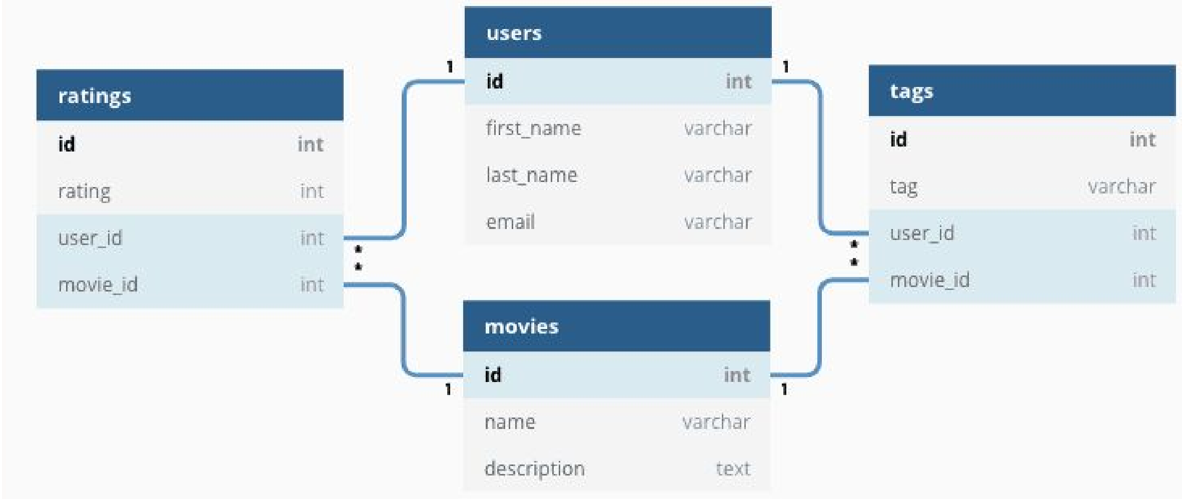

fig.2.3.7

9.Cloud Databases:

Nowadays, data has been specifically getting stored over clouds also known as a virtual environment, either in a hybrid cloud, public or private cloud. A cloud database is a database that has been optimized or built for such a virtualized environment. There are various benefits of a cloud database, some of which are the ability to pay for storage capacity and bandwidth on a per-user basis, and they provide scalability on demand, along with high availability.

A cloud database also gives enterprises the opportunity to support business applications in a software-as-a-service deployment.

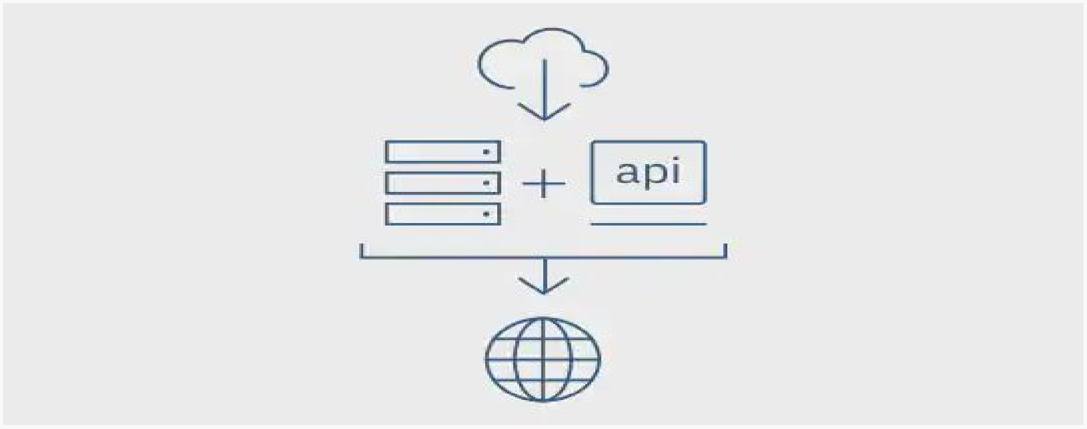

fig.2.3.8

10.Object-Oriented Databases:

An object-oriented database is a collection of object-oriented programming and relational databases. There are various items which are created using object-oriented programming languages like C++, Java which can be stored in relational databases, but object-oriented databases are well-suited for those items.

An object-oriented database is organized around objects rather than actions, and data rather than logic. For example, a multimedia record in a relational database can be a definable data object, as opposed to an alphanumeric value.

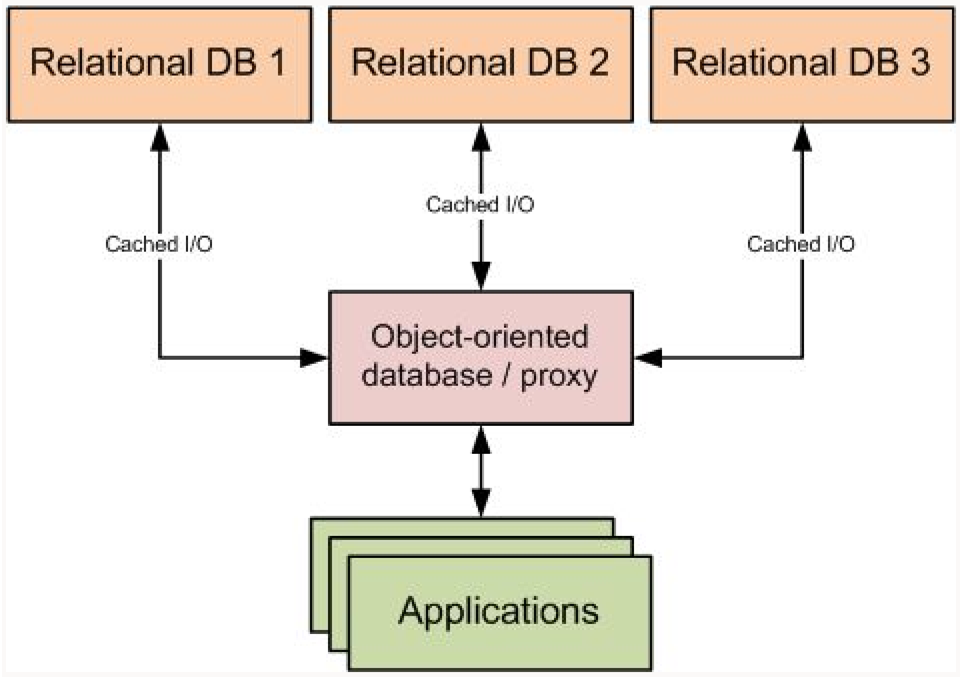

fig.2.3.9

11.Graph Databases:

The graph is a collection of nodes and edges where each node is used to represent an entity and each edge describes the relationship between entities. A graph-oriented database, or graph database, is a type of NoSQL database that uses graph theory to store, map and query relationships.

Graph databases are basically used for analyzing interconnections. For example, companies might use a graph database to mine data about customers from social media.

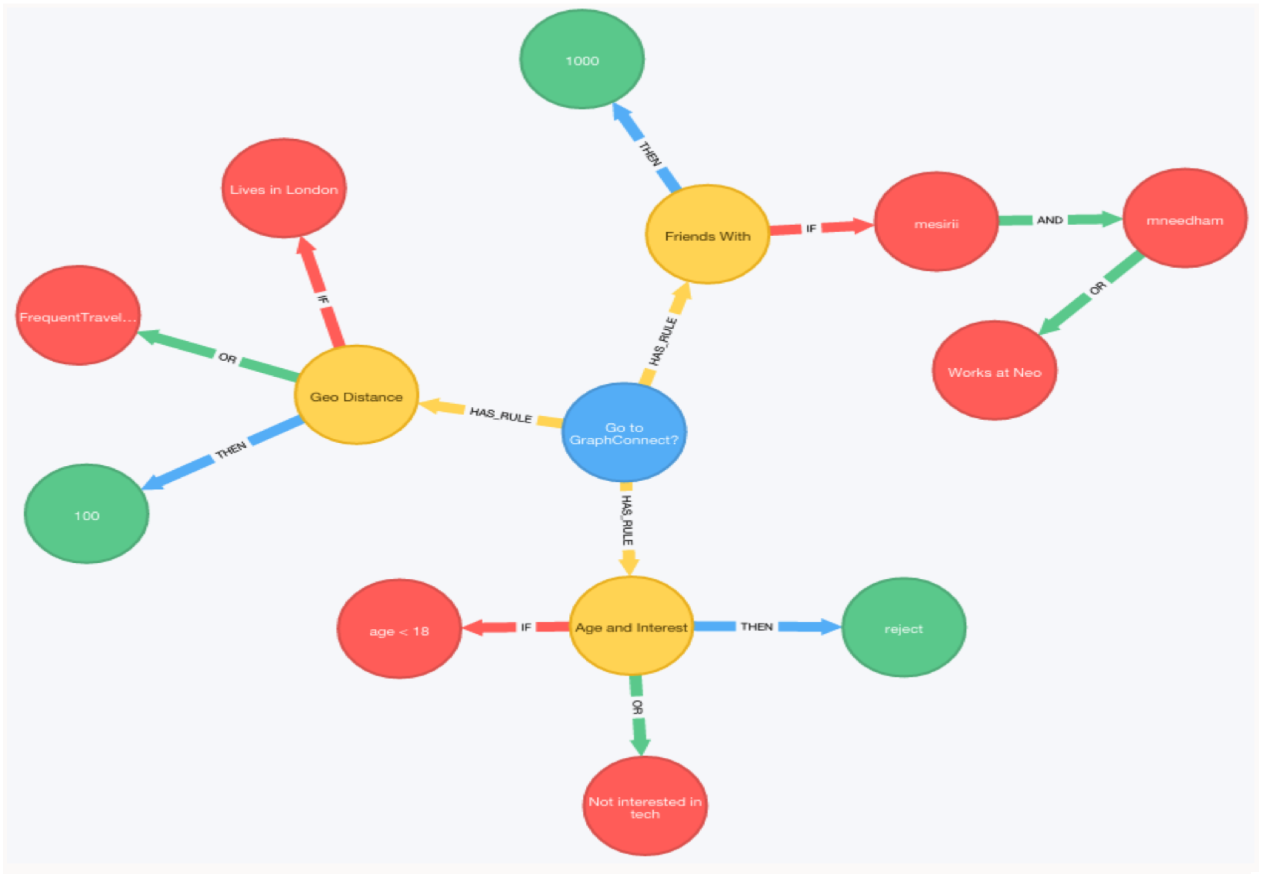

fig.2.3.10

2.4 Node.Js

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009 and its latest version is v0.10.36. The definition of Node.js as supplied by its official documentation is as follows −

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

Features of Node.js

Following are some of the important features that make Node.js the first choice of software architects.

- Asynchronous and Event Driven − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
- Very Fast − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- Single Threaded but Highly Scalable − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- No Buffering − Node.js applications never buffer any data. These applications simply output the data in chunks.
- License − Node.js is released under the MIT license

CHAPTER-3

REQUIREMENTS

3.1SOFTWARE REQUIREMENTS:

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

NUMBER	DESCRIPTION
1.Operating system	Windows 7 or later
2.Languages used	SQL,Query,Database,Node.js
3.compiler	Visual studio code

table.3.1

3.2 Hardware requirements:

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following subsections discuss the various aspects of hardware requirements.

Processor	Description
RAM	4.0GB(minimum)
Hard disk drive	400 GB(minimum)

Table.3.2

CHAPTER-4
ABOUT PROJECT

4.1 Project Name:MEDICINO (BACKEND DEVELOPMENT)

Our project name is MEDICINO which is on hospital management. which provides separate portals for doctors, patients,nurse,receptionists and admin.every portal consists of profile updation,attendance updation and dashboard except in admin and patient portal.patient portal consists of patient information page and admin portal consists of profile updation and reports regarding patients.

4.2 Technologies used:
1.SQL
2.QUERY
3.Database
4.Node.js

4.3 Technical details:

1. HTML,CSS ,JS,JQuery,Bootstrap are used to design the front end part.

3. Visual studio code is used for compilation.

CHAPTER-5
MODULES

5.1 User registration:

The registration page is a special page where users can create an account on your site. The theme extends the registration form with social networks registration .registration page is for new users of the hospital.

Our registration page consists of following fields:

1.email-id
2.Username
3.Password
4.Date of birth
5.Mobile number
6.Address

These fields are done with validations by using regular expressions
Email-id:

```
var em  =/^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/;
```

User name:

```
            var us = /^[A-Z][a-z A-Z]{6,12}$/;
```

password:

```
             var pas = /^[a-z A-Z 0-9 ! @ # $]{6,12}$/;
```

Date of birth:

```
            var dob = /^(0?[1-9]|[12][0-9]|3[01])[\/\-](0?[1-9]|1[012])[\/\-]\d{4}$/;
```

Mobile number:

```
             var num = /^[6-9][0-9]{9}$/;
```

5.2 Login Page:

The login page allows a user to gain access to an application by entering their username and password after registration.

Login page consists of following fields:

1.Username
2.Password

These are done with validations by using regular expressions

User name:

```
            var us = /^[A-Z][a-z A-Z]{6,12}$/;
```

password:

```
             var pas = /^[a-z A-Z 0-9 ! @ # $]{6,12}$/;
```

CHAPTER-6
Codes

Database code:

```
create database db2;
use db2;
 create table registers(
  doctorid int auto_increment,
 username varchar(100),
  emailid varchar(100),
 mobilenumber bigint,
 password1 varchar(20),
address varchar(200),
primary key(doctorid)
);
select *from registers;
create table reception(
 receptionid int auto_increment,
 username varchar(100),
  emailid varchar(100),
 mobilenumber bigint,
 password1 varchar(20),
address varchar(200),
primary key(receptionid)
);
create table patient(
 patientid int auto_increment,
 username varchar(100),
  emailid varchar(100),
 mobilenumber bigint,
 password1 varchar(20),
address varchar(200),
primary key(patientid)
);
create table nurses(
 nurseid int auto_increment,
 username varchar(100),
  emailid varchar(100),
 mobilenumber bigint,
 password1 varchar(20),
address varchar(200),
primary key(nurseid)
);
create table admin(
 adminid int auto_increment,
 username varchar(100),
  emailid varchar(100),
 mobilenumber bigint,
 password1 varchar(20),
address varchar(200),
primary key(adminid)
);

 create table reports(
 patientid int auto_increment,
 patientname varchar(30),
 mobilenumber bigint,
 address varchar(200),
 status varchar(200),
 primary key(patientid)
 );
  create table login(
 id int auto_increment,
 username varchar(30),
email varchar(100),
 mobilenumber bigint,
 address varchar(200),
  password varchar(20),
 primary key(id)
 );
 select * from nurseupd;
  create table recept(
 name varchar(30),
 feedback varchar(50)
 );
 create table appointment(
 patientid int auto_increment,
 name varchar(100),
  email varchar(100),
 mobilenumber bigint,
address varchar(200),
primary key(patientid)
);
create table updation(
name varchar(30),
feedback varchar(50)
);
create table attendance(
name varchar(30),
feedback varchar(50)
);
create table nurseupd(
name varchar(30),
feedback varchar(50)
);
select * from login;
show tables;
select * from patient;
select * from attendance;
select * from updation;
select * from nurseupd;
```

Code for config:

```
module.exports = {
  HOST: "localhost",
  USER: "root",
  PASSWORD: "dsps@123",
  DB: "db2",
};
```

Controllers code:
Admin controller code:

```
const Admin = require("../models/admin.model.js");



exports.create = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content can not be empty!"
        });
    }

    const admin = new Admin({
        username:req.body.username,
        emailid: req.body.emailid,
        mobilenumber: req.body.mobilenumber,
        password1: req.body.password1,
        address:req.body.address,
    });

    Admin.create(admin, (err, data) => {
        if (err)
            res.status(500).send({
                message:
                    err.message || "Some error occurred while creating the admin."
            });
```

```
      else res.send(data);
  });
};
```

Appointment controller code:

```
const Appointment = require("../models/appointment.model.js");


exports.create = (req, res) => {
  // Validate request
  if (!req.body) {
    res.status(400).send({
      message: "Content can not be empty!"
    });
  }

  const appointment = new Appointment({
    name:req.body.name,
    email: req.body.email,
    mobilenumber: req.body.mobilenumber,
    address:req.body.address,
  });

  Appointment.create(appointment, (err, data) => {
    if (err)
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the Appointment."
      });
    else res.send(data);
  });
};
```

Attendance controller code:

```
const Attendanced = require("../models/attendanced.model.js");
```

```
exports.create = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content can not be empty!"
        });
    }

    const attendanced = new Attendanced({
        name:req.body.name,
        feedback: req.body.feedback,
    });

    Attendanced.create(attendanced, (err, data) => {
        if (err)
            res.status(500).send({
                message:
                    err.message || "Some error occurred while creating the attendance."
            });
        else res.send(data);
    });
};
```

Login controller code:

```
const Logins = require("../models/login.model.js");



exports.create = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content can not be empty!"
        });
    }
    const logins = new Logins({
        username:req.body.username,
        email: req.body.email,
        dateofbirth:req.body.dateofbirth,
        mobilenumber: req.body.mobilenumber,
        address:req.body.address,
        password: req.body.password,
    });

    Logins.create(logins, (err, data) => {
        if (err)
            res.status(500).send({
                message:
                    err.message || "Some error occurred while creating the login."
            });
        else res.send(data);
    });
};
```

Nurse controller code:

```
const Nurse = require("../models/nurse.model.js");



exports.create = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content can not be empty!"
        });
    }

    const nurse = new Nurse({
        username:req.body.username,
        emailid: req.body.emailid,
        mobilenumber: req.body.mobilenumber,
      password1: req.body.password1,
      address:req.body.address,
   });

   Nurse.create(nurse, (err, data) => {
      if (err)
         res.status(500).send({
            message:
               err.message || "Some error occurred while creating the Nurse."
         });
      else res.send(data);
   });
};
```

Patient controller code:

```
const Patient = require("../models/patient.model.js");

exports.create = (req, res) => {
  // Validate request
  if (!req.body) {
   res.status(400).send({
     message: "Content can not be empty!"
   });
  }

  const patient = new Patient({
   username:req.body.username,
   emailid: req.body.emailid,
   mobilenumber: req.body.mobilenumber,
   password1: req.body.password1,
   address:req.body.address,
  });

  Patient.create(patient, (err, data) => {
   if (err)
     res.status(500).send({
      message:
        err.message || "Some error occurred while creating the patient."
    });
   else res.send(data);
 });
};
```

Reception controller code:

```
const Reception = require("../models/reception.model.js");

exports.create = (req, res) => {
  // Validate request
  if (!req.body) {
    res.status(400).send({
      message: "Content can not be empty!"
    });
  }

  const reception = new Reception({
    username:req.body.username,
    emailid: req.body.emailid,
    mobilenumber: req.body.mobilenumber,
    password1: req.body.password1,
    address:req.body.address,
  });

  Reception.create(reception, (err, data) => {
    if (err)
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the
reception."
      });
    else res.send(data);
  });
};
```

Register controller code:

```
const Register = require("../models/register.model.js");


// Create and Save a new Register
exports.create = (req, res) => {
  // Validate request
  if (!req.body) {
    res.status(400).send({
      message: "Content can not be empty!"
    });
  }

  // Create a Register
  const register = new Register({
    username:req.body.username,
    emailid: req.body.emailid,
    mobilenumber: req.body.mobilenumber,
    password1: req.body.password1,
    address:req.body.address,
  });

  // Save Register in the database
  Register.create(register, (err, data) => {
    if (err)
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the Register."
      });
    else res.send(data);
  });
};
```

Reports controller code:

```
const Reports = require("../models/report.model.js");

exports.create = (req, res) => {
  // Validate request
  if (!req.body) {
    res.status(400).send({
      message: "Content can not be empty!"
    });
  }

  const reports = new Reports({
    patientname:req.body.patientname,
    mobilenumber: req.body.mobilenumber,
    address:req.body.address,
    status:req.body.status,
  });

  Reports.create(reports, (err, data) => {
    if (err)
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the Reports."
      });
    else res.send(data);
  });
};
```

Updation controller:

```
const Updation = require("../models/updation.model.js");


exports.create = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content can not be empty!"
        });
    }

    const updation = new Updation({
        name:req.body.name,
        feedback: req.body.feedback,
    });

    Updation.create(updation, (err, data) => {
        if (err)
            res.status(500).send({
                message:
                    err.message || "Some error occurred while creating the updation."
            });
        else res.send(data);
    });
};
```

MODELS:

Admin:

```
const sql = require('../models/db.js');

const Admin = function (admin) {
    this.username = admin.username;
    this.emailid = admin.emailid;
    this.mobilenumber = admin.mobilenumber;
    this.password1 = admin.password1;
    this.address = admin.address;
};

Admin.create = (newadmin, result) => {
    sql.query(`insert into admin set ?`, newadmin, (err, res) => {
        if (err) {
            console.log(err);
            result(err, null);
            return;
        }
        console.log("Created admin : ", { id: res.insertedId, ...newadmin });
        return (null, { id: res.insertedId, ...newadmin });
    })
};
module.exports = Admin;
```

Appointment:

```
const sql = require('../models/db.js');

const Appointment = function(appointment){
    this.name = appointment.name;
    this.email = appointment.email;
    this.mobilenumber = appointment.mobilenumber;
    this.address=appointment.address;
};

Appointment.create = (newAppointment,result) => {
    sql.query('insert into appointment set ?',newAppointment,(err,res) =>{
        if(err){
            console.log(err);
            result(err,null);
            return;
        }
        console.log("Created appointment : ",{id:res.insertedId,...newAppointment});
        return (null,{id:res.insertedId,...newAppointment});
    })
};

module.exports = Appointment;
```

Attendance:

```
const sql = require('../models/db.js');

const Attendanced = function(attendanced){
    this.name = attendanced.name;
    this.feedback = attendanced.feedback;
};

Attendanced.create = (newAttendanced,result) => {
    sql.query('insert into attendance set ?',newAttendanced,(err,res) =>{
        if(err){
            console.log(err);
            result(err,null);
            return;
        }
        console.log("Created attendance : ",{id:res.insertedId,...newAttendanced});
        return (null,{id:res.insertedId,...newAttendanced});
    })
};




module.exports = Attendanced;
```

Database:

```
const mysql = require('mysql');
const dbConfig = require('../config/db.config.js');
const connection = mysql.createConnection({
    host: dbConfig.HOST,
    user:dbConfig.USER,
    password:dbConfig.PASSWORD,
    database:dbConfig.DB
});

connection.connect(error =>{
    if(error){
        return console.error(error.message);
    }
    console.log('Successfully connected to MySQL Database');
});

module.exports = connection;
```

Login:

```
const sql = require('../models/db.js');


const Logins = function(logins){
    this.username = logins.username;
    this.email = logins.email;
    this.dateofbirth=logins.dateofbirth;
    this.mobilenumber = logins.mobilenumber;
    this.address=logins.address;
    this.password=logins.password;
};


Logins.create = (newLogins,result) => {
    sql.query('insert into login set ?',newLogins,(err,res) =>{
        if(err){
            console.log(err);
            result(err,null);
            return;
        }
        console.log("Created login : ",{id:res.insertedId,...newLogins});
        return (null,{id:res.insertedId,...newLogins});
    })
};





module.exports = Logins;
```

Nurse:

```
const sql = require('../models/db.js');

const Nurse = function(nurse){
    this.username = nurse.username;
    this.emailid = nurse.emailid;
    this.mobilenumber = nurse.mobilenumber;
    this.password1=nurse.password1;
    this.address=nurse.address;
};

Nurse.create = (newNurse,result) => {
    sql.query('insert into nurses set ?',newNurse,(err,res) =>{
        if(err){
            console.log(err);
            result(err,null);
            return;
        }
        console.log("Created Nurse : ",{id:res.insertedId,...newNurse});
        return (null,{id:res.insertedId,...newNurse});
    })
};




module.exports = Nurse;
```

Patient:

```
const sql = require('../models/db.js');


const Patient = function(patient){
    this.username = patient.username;
    this.emailid = patient.emailid;
    this.mobilenumber = patient.mobilenumber;
    this.password1=patient.password1;
    this.address=patient.address;
};
```

```
Patient.create = (newPatient,result) => {
    sql.query('insert into patient set ?',newPatient,(err,res) =>{
        if(err){
            console.log(err);
            result(err,null);
            return;
        }
        console.log("Created patient : ",{id:res.insertedId,...newPatient});
        return (null,{id:res.insertedId,...newPatient});
    })
};

module.exports = Patient;
```

Receptionist:

```
const sql = require('../models/db.js');


const Reception = function(reception){
    this.username = reception.username;
    this.emailid = reception.emailid;
    this.mobilenumber = reception.mobilenumber;
    this.password1=reception.password1;
    this.address=reception.address;
};


Reception.create = (newReception,result) => {
    sql.query('insert into reception set ?',newReception,(err,res) =>{
        if(err){
            console.log(err);
            result(err,null);
            return;
        }
        console.log("Created receptionist : ",{id:res.insertedId,...newReception});
        return (null,{id:res.insertedId,...newReception});
    })
};

module.exports = Reception;
```

Registration:

```
const sql = require('../models/db.js');

const Register = function(register){
    this.username = register.username;
    this.emailid = register.emailid;
    this.mobilenumber = register.mobilenumber;
    this.password1=register.password1;
    this.address=register.address;
};

Register.create = (newRegister,result) => {
    sql.query('insert into registers set
?',newRegister,(err,res) =>{
        if(err){
            console.log(err);
            result(err,null);
            return;
        }
        console.log("Created Register :
",{id:res.insertedId,...newRegister});
        return (null,{id:res.insertedId,...newRegister});
    })
};

module.exports = Register;
```

Reports:

```
const sql = require('../models/db.js');

const Reports = function(reports){
   this.patientname = reports.patientname;
   this.mobilenumber = reports.mobilenumber;
```

```
    this.address=reports.address;
    this.status = reports.status;

};

Reports.create = (newReports,result) => {
    sql.query('insert into Reports set ?',newReports,(err,res) =>{
        if(err){
            console.log(err);
            result(err,null);
            return;
        }
        console.log("Created Report : ",{id:res.insertedId,...newReports});
        return (null,{id:res.insertedId,...newReports});
    })
};

module.exports = Reports;
```

Updation:

```
const sql = require('../models/db.js');

const Updation = function(updation){
    this.name = updation.name;
    this.feedback = updation.feedback;
};

Updation.create = (newupdation,result) => {
    sql.query('insert into updation set ?',newupdation,(err,res) =>{
        if(err){
            console.log(err);
            result(err,null);
            return;
        }
        console.log("Created updation : ",{id:res.insertedId,...newupdation});
        return (null,{id:res.insertedId,...newupdation});
    })
};



module.exports = Updation;
```

ROUTES:

Admin:

```
module.exports = app =>{
    const admins = require('../controllers/admin.controller.js');


    app.post ("/admins",admins.create);


}
```

Appointment:

```
module.exports = app =>{
    const appointment =
require('../controllers/appointment.controller.js');


    app.post ("/appointment",appointment.create);


}
```

Attendance:

```
module.exports = app =>{
    const attendenced = require('../controllers/attendanced.controller.js');


    app.post ("/attendence",attendenced.create);


}
```

Login:

```
module.exports = app =>{
    const logins =
require('../controllers/login.controller.js');


    app.post ("/login",logins.create);


}
```

Nurse:

```
module.exports = app =>{
    const nurses =
require('../controllers/nurse.controller.js');


    app.post ("/nurses",nurses.create);


}
```

Patient:

```
module.exports = app =>{
    const patients = require('../controllers/patient.controller.js');


    app.post ("/patient",patients.create);


}
```

Receptionist:

```
module.exports = app =>{
    const reception = require('../controllers/reception.controller.js');
        app.post ("/reception",reception.create);
    }
```

Register:

```
module.exports = app =>{
    const register = require('../controllers/register.controller.js');
    //create a new register
    app.post ("/register",register.create);


}
```

Report:

```
module.exports = app =>{
    const reports = require('../controllers/report.controller.js');

    app.post ("/reports",reports.create);


}
```

Updation:

```
module.exports = app =>{
    const updation = require('../controllers/updation.controller.js');



    app.post ("/updation",updation.create);


}
```

CHAPTER-7
MAINTENANCE

Project maintenance:

The process of tracking and enabling project activities in accordance with the project plan is an essential, but often overlooked, factor in overall project success. After spending so much time in the planning phase, many project managers have a tendency to take a step back once the other members of the project team start their work, but experienced project managers know that project monitoring is every bit as important as project planning.

Enterprise projects, in particular, require a steady commitment to project maintenance, simply because they tend to have much longer durations than projects undertaken at smaller organizations. The longer a project runs, the more likely it becomes that a small deviation from the project plan will snowball into a serious issue as the project progresses. No matter what size projects you’re managing today, an appreciation for project maintenance can only improve your chances of success. These strategies below can help you keep your eye on the prize even during the longest-running projects.

For maintenance of the website:

1. The database has to be updated regularly according to new available information.
2. Redundant and false information must be removed from the database.
3. Newer versions of PHP and MYSQL can be used for upgradation of web sites And to improve the overall performance of the system.

CHAPTER-8
FUTURE SCOPE AND FUTURE ENHANCEMENT

This project(hospital management)helps in knowing of information about hospital regarding doctors and patients.The admin of the hospital can easily know the information about ratio of the doctors and patients,doctors attendance,new joinings of hospital staff.It helps to know about the patients information too where admin can easily calculate no.of beds, machinery required for patients treatment and no.of patients recovered.

Project name:

MEDICINO(Hospital management)

CHAPTER-9
CONCLUSION

We have successfully implemented the site ‘MEDICINO’ with the help of various technologies and tools,we have been able to provide a site which will be live soon and running on the web. We have been successful in our attempt to take care of the needs of both the user as well as the administrator.Finally we hope that this will go a long way in popularizing.