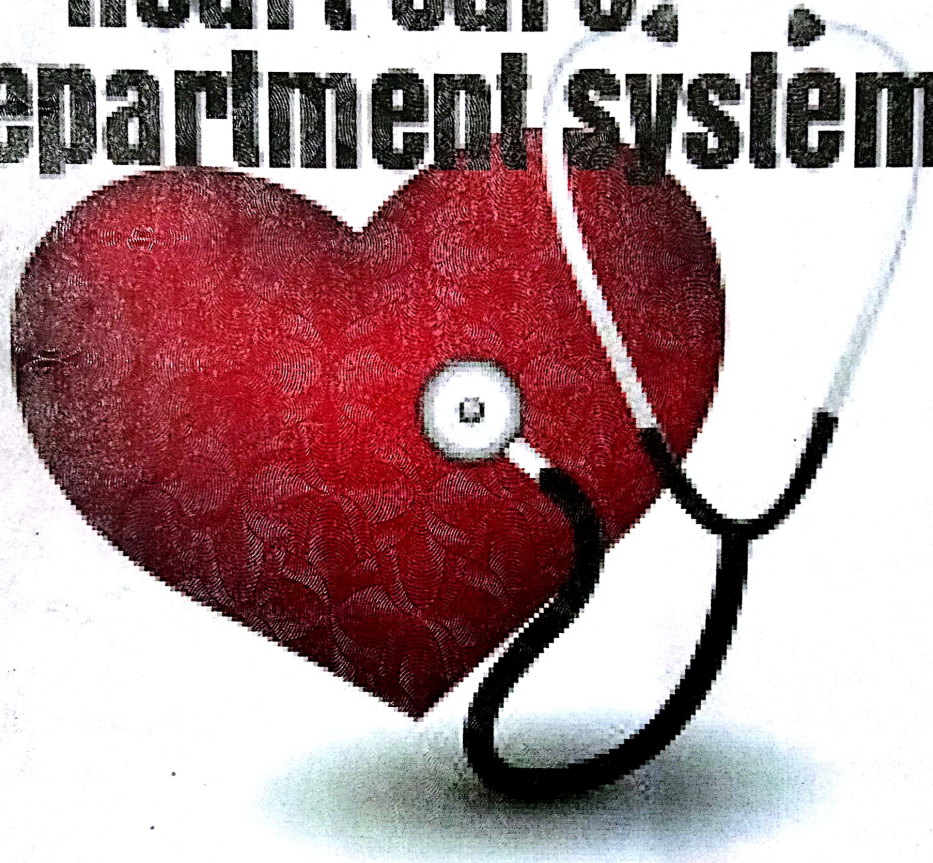
Heart care department system



Dr. Ahmed Salah
Eng. Mahmoud Mahdi

Heart Care Department System

Dr. Ahmed Salah

Eng. Mahmoud Mahdi

Group Names

Aya Hosny

Fatma El Sayed

ESraa Abdel basset

Al Shimaa Abdel Fatah

Department of computer science

TABLE OF CONTENTS

INTRODUCTION

MOTIVATION

Background of the project idea

The idea is that the heart department in the university hospital is divided into two parts, one branch and the other in the university hospital. Patients' data was recorded in paper documents that are difficult to follow. Here is the problem that doctors who manage the two branches together cannot follow up at the same time. For this problem, we proposed a system that combines the two branches together and they are one branch and thus easy to carry with their luggage easily and follow-up patients first, wherever the doctor concerned and thus saved the effort and time and loss of patient data resulting from recording their data in paper sheets can be damaged and lost

Problem

Heart care department is still using paper sheets to record the patient conditions and updates and it has many disadvantages

- Consuming a lot of papers
- Papers could get lost or cut
- It is hard to keep archive of all patients ever been to the hospital because it means so many papers
- It is hard for doctors and supervisors to mentor the activities of the nursing team through their paper work as it lakes to time factor and anyone can fill it

Our system provides **solution** to all these problems in addition to many other features that accelerates the processes that happen in the ICU and guarantees data integrity and security.

Data Sources

Interview with Dr. Ahmed Taha and his team in the university hospital.

CHAPTER III: METHODOLOGY

SECTION 1: MAIN WORK

We can divide our application's main work into main four parts:

1. Meeting the heart care department team and note how their paper based system worked and what do they expect from the new system
2. learning web developing
3. We had to learn web development all over again to make sure out knowledge is sufficient enough to develop such project

   We learned:

   - HTML
   - CSS
   - JavaScript
   - Angular.js
   - Mongo
   - Node.js
4. Determining the tools will be used to develop the project

SECTION 2: APPLICATIONS AND LANGUAGES USED

Programming Languages

1. HTML
2. CSS
3. Java script

Applications

1. Framework meteor
2. Visual studio
3. Mongo Database

HTML

Hypertext Markup Language, commonly abbreviated as HTML, is the standard markup language used to create web pages. Along with CSS, and JavaScript, HTML is a cornerstone technology used to create web pages, as well as to create user interfaces for mobile and web applications. Web browsers can read HTML files and render them into visible or audible web pages. HTML describes the structure of a website semantically and, before the advent of Cascading Style Sheets (CSS), included cues for the presentation or appearance of the document (web page), making it a markup language, rather than a programming language.

HTML elements form the building blocks of HTML pages. HTML allows images and other objects to be embedded and it can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> introduce content into the page directly. Others such as ... surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language. Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document, including plain XML, SVG and XUL, and is applicable to rendering in speech, or on other media. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites

to create visually engaging webpages, user interfaces for web applications, and user interfaces for many mobile applications.

CSS is designed primarily to enable the separation of document content from document presentation, including aspects such as the layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

METEOR

Meteor is a full-stack JavaScript platform for developing modern web and mobile applications. Meteor includes a key set of technologies for building connected-client reactive applications, a build tool, and a curated set of packages from the Node.js and general JavaScript community. and it is a collection of libraries and packages that are bound together in a tidy way to make web development easier. It builds on ideas from previous frameworks and libraries to offer an easy way to start a prototype app, but it gives you the tools and flexibility to build a full-fledged production app. There are libraries like Angular and Blaze

- Meteor allows you to develop in one language, JavaScript, in all environments: application server, web browser, and mobile device.
- Meteor uses data on the wire, meaning the server sends data, not HTML, and the client renders it.
- Meteor supports OS X, Windows, and Linux.

In a way, it is. Meteor has been built on concepts from other frameworks and libraries in a way that makes it easy to prototype applications. Essentially, it makes web development easier. It's flexible and requires less code, which means less bugs and typically a higher quality and more stable end result.

Meteor is easy to learn and quick to build with—making it a new favorite for many developers. So what are the top things that make this framework a favorite? Let's take a look at the best reasons to build with Meteor.

8 Reasons to Build with Meteor

1. its fast to build with.
2. Develop with just one language.
3. Easy to learn.
4. Real-time applications at the core.
5. Smart packages that save you time.
6. Easily convert to native mobile apps.
7. Active and supportive community.
8. Meteor is the future.

1. Its fast to build with.

When using Meteor, launching a MVP in 3-4 weeks can be a reality. With JavaScript on the front-end and back-end, plus smart packages, Meteor allows you to develop faster. This makes it a go-to for startups and others trying to get a product out the door quickly.

2. Develop with just one language.

Aside from rapid development, JavaScript on the server and client has other benefits—like less context switching. Moreover, JavaScript is one of the most popular programming languages in the world. It has many use-cases, and many developers are already familiar with it.

3. Easy to learn.

Anyone who gives Meteor a try knows how simple it is to get up and going. Unlike other popular full stack frameworks, you don't have to rely on multiple languages. Any semi-experienced JS developer could be handed a Meteor project with decent structure, and pick it up and get it running quickly. Even those who are not

full-fledged developers, like designers, find it easy to implement. They love it for the same reasons as everyone: it's simple to learn and work with.

4. Real-time applications at the core.

Meteor is the perfect solution for those looking to build real-time applications. It is real-time by default, known as "full stack reactivity". All of the application's layers from database to template update automatically. This means there is no need to refresh the page to see updates. And any changes to documents save instantly. This makes Meteor a perfect use-case for real time collaboration, too.

5. Smart packages that save you time.

Building out a login system for your application can be a pain. But not with Meteor. Meteor packages make it simple to add features such as:

- User accounts
- JavaScript libraries like React
- Extras like Bootstrap or Stylus and more

There is even an entire website dedicated to Meteor package management. Even better, adding these kinds of smart packages is easy: just a few keystrokes in the terminal.

6. Turn your Meteor app into iOS and Android apps.

Yes, you read that right. Meteor makes it simple to turn your web app into a smartphone app with Cordova. Cordova is a platform to build native smartphone applications using HTML, CSS and JavaScript. It has a set of APIs that allow you to access native device functions, like the camera, with JavaScript. And the best part is that Meteor comes out of the box with Cordova.

7. Active and supportive community.

Despite its young age, there is already a passionate community of web developers using Meteor—"Meteorites", as they call themselves. More than that, tons of

blogs, resources, and even online learning platforms have emerged to discuss and teach the platform. Meteor's popularity can be seen by fact that it is the 10th most starred repository on Github, with 27,085 stars at the time of writing. There is also a large, and constantly growing, in-person community. There are Meteor Meetups in over 75 countries around the world, and in 217 cities. Chances are there is a Meteor Meetup near you.

8. Meteor is the future.

The web is becoming an increasingly real-time environment. With the growth of online applications like real-time collaboration tools, instant customer support, multiplayer web games, and more, the need for real-time friendly applications is increasing. And Meteor offers just that.

Meteor—A Bright Future

Meteor is still in its early days. But it is constantly evolving and being improved. However, a few things are certain:

Meteor is fast and easy to implement—less code and rapid development makes it fun to build with—for experienced devs and those just starting out

Real-time applications are the present and future—Meteor makes it easy to build real-time apps on desktop and mobile devices

Meteor has an amazing community

JAVASCRIPT

JavaScript is a high-level, dynamic, untipped, and interpreted programming language. It has been standardized in the ECMAScript language specification. Alongside HTML and CSS, it is one of the three core technologies of World Wide Web content production; the majority of websites employ it and it is supported by all modern Web browsers without plug-ins. JavaScript is prototype-based with

first-class functions, making it a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles. It has an API for working with text, arrays, dates and regular expressions, but does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.

Although there are strong outward similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two are distinct languages and differ greatly in their design. JavaScript was influenced by programming languages such as self and Scheme.

MongoDB

MongoDB (from humongous) is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas. MongoDB is developed by MongoDB Inc. and is free and open-source, published under a combination of the GNU Affero General Public License and the Apache License.

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

A document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	Mongod
mysql/sqlplus	Mongo

TABLE 1

To run Mongo you must do that:

1. Open command prompt
2. Copy meteor mongo

After open session go and copy your code and mongo create column and table by itself.

Import mongo in code API

```
import { Mongo } from 'meteor/mongo';
```

NODE.JS

Open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side.

ANGULAR.JS

Angular is a JavaScript-based open-source front-end web application framework mainly maintained by Google and by a community of individuals and corporation to address many of the challenges encountered in developing single-page applications. The JavaScript components complement Apache Cordova, the framework used for developing cross-platform mobile apps. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–view model (MVVM) architectures, along with components commonly used in rich Internet applications. In 2014, the original AngularJS team began working on Angular (Application Platform).

The AngularJS framework works by first reading the HTML page, which has embedded into it additional custom tag attributes. Angular interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The values of those JavaScript variables can be manually set within the code, or retrieved from static or dynamic JSON resources.

According to JavaScript analytics service Libscore, AngularJS is used on the websites of Wolfram Alpha, NBC, Walgreens, Intel, Sprint, ABC News, and approximately 12,000 other sites out of 1 million tested in October 2016.

AngularJS is the frontend part of the MEAN stack, consisting of MongoDB database, Express.js web application server framework, Angular.js itself, and Node.js server runtime environment.

AngularJS is built on the belief that declarative programming should be used to create user interfaces and connect software components, while imperative programming is better suited to defining an application's business logic. The framework adapts and extends traditional HTML to present dynamic content through two-way data-binding that allows for the automatic synchronization of models and views. As a result, Angular.JS de-emphasizes explicit DOM manipulation with the goal of improving testability and performance.

AngularJS's design goals include:

- To decouple DOM manipulation from application logic. The difficulty of this is dramatically affected by the way the code is structured.
- To decouple the client side of an application from the server side. This allows development work to progress in parallel, and allows for reuse of both sides.
- To provide structure for the journey of building an application: from designing the UI, through writing the business logic, to testing.

SETTING UP A WORK PLAN TO KEEP OUR WORK ORGANIZED

SYSTEM ANALYSIS

Use-Case

The project use case diagram

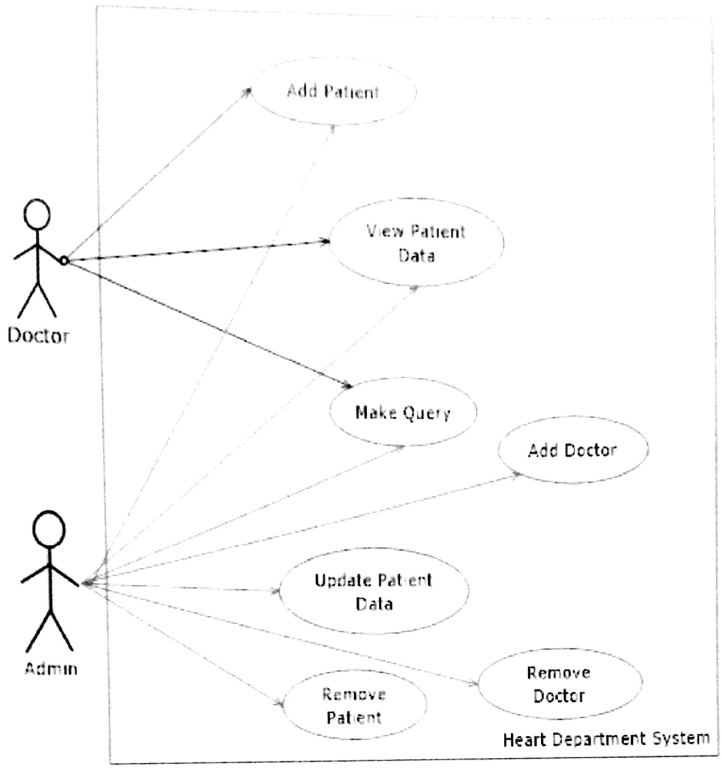


FIGURE 1: SYSTEM USE CASE

In this system there are two actors the first one is the Doctor and the other one is the Admin

The use cases for each actor

- The Doctor
  1. Add Patient
  2. View Data
  3. Make a query
- The Admin
  1. Add Patient
  2. View Data
  3. Make a query
  4. Add Doctor

5. Update Patient Data
6. Remove Patient
7. Remove Doctor

THE SYSTEM ERD

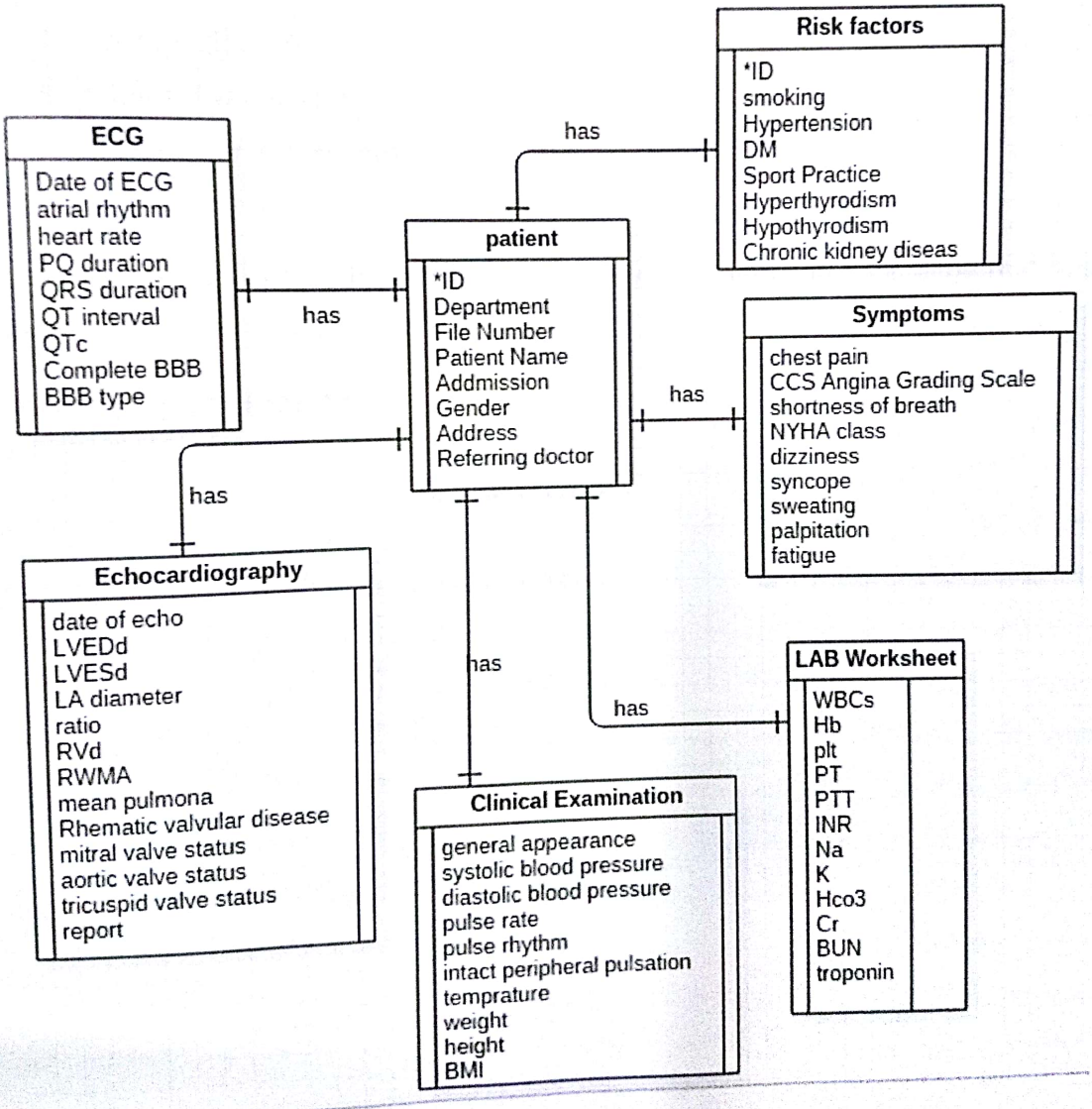


FIGURE 2: SYSTEM ERD DIAGRAM

Now we convert our project from the level of analysis to ERD, we have the main tables:

1. ECG
1. Patient
2. risk factors
3. Symptoms
4. Echocardiography
5. Clinical Examination
6. Laboratory Worksheet

These is our tables that have relationship between them and core table that has relation with most table is **patient**

THE SEQUENCE DIAGRAM

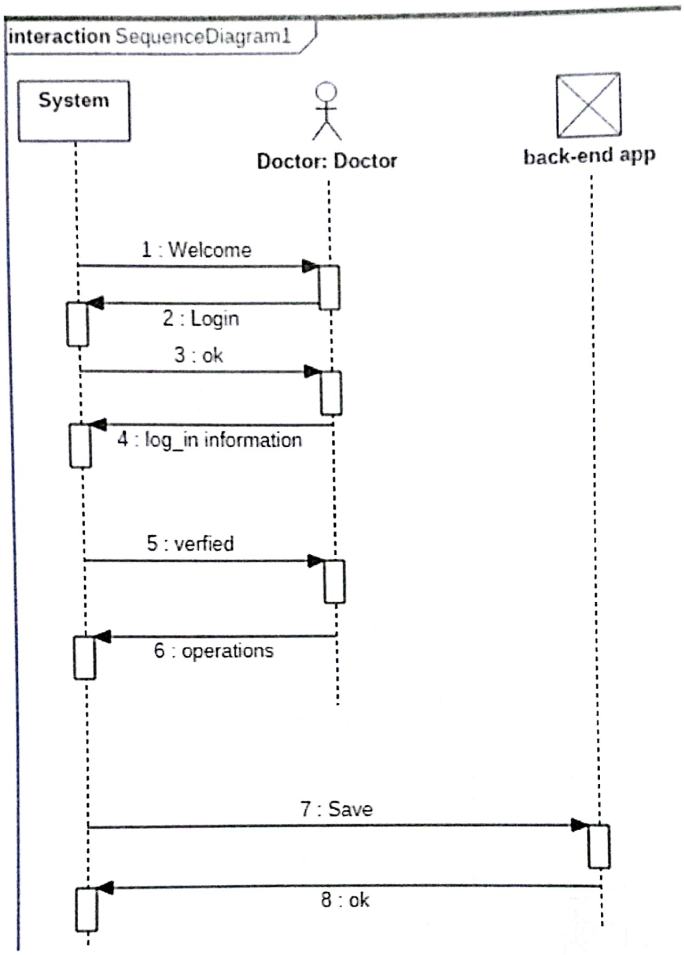


FIGURE 3: SEQUENCE DIAGRAM

THE COMPONENT DIAGRAM

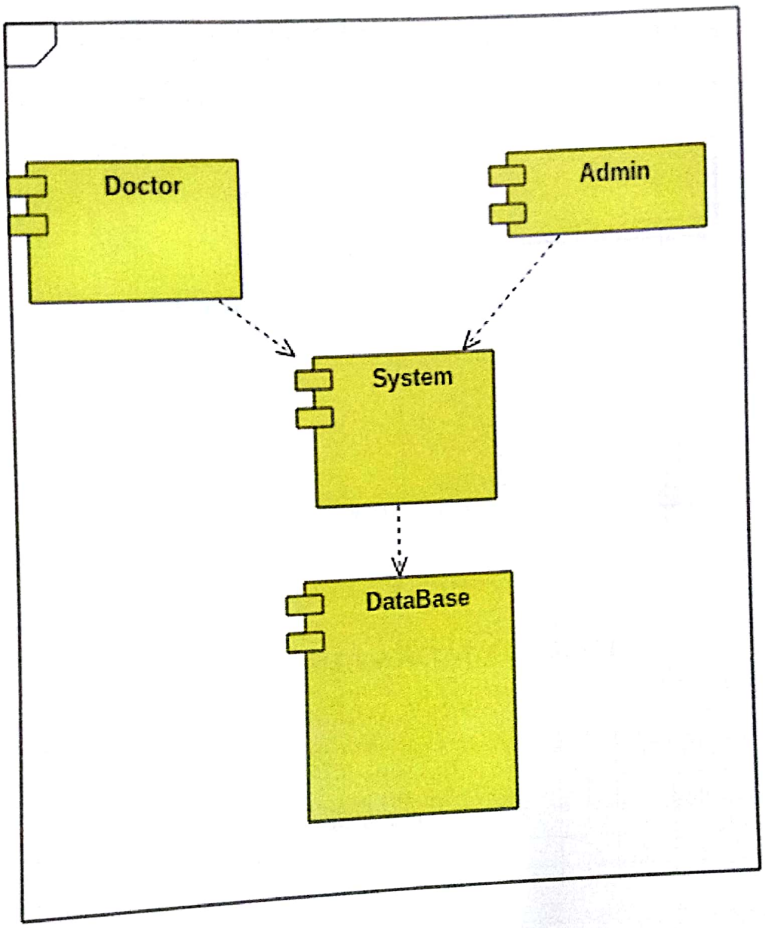


FIGURE 4 : SYSTEM COMPONENT DIAGRAM

THE SYSTEM ACTIVITY DIAGRAM

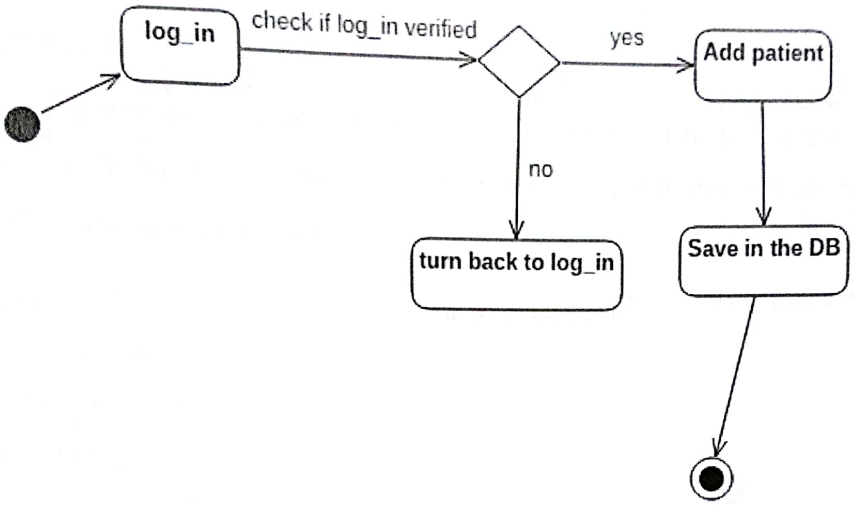


FIGURE 5 : SYSTEM ACTIVITY DIAGRAM

SYSTEM DESIGN

USER INTERFACE

User interface design is an art. The goal is to make the interface pleasing to the eyes and simple to use, while minimize the effort users expand to accomplish their work. These principals are:

1) Layout
2) Content awareness
3) Aesthetics
4) Consistency
5) Minimize user effort

Examples of Screen Design

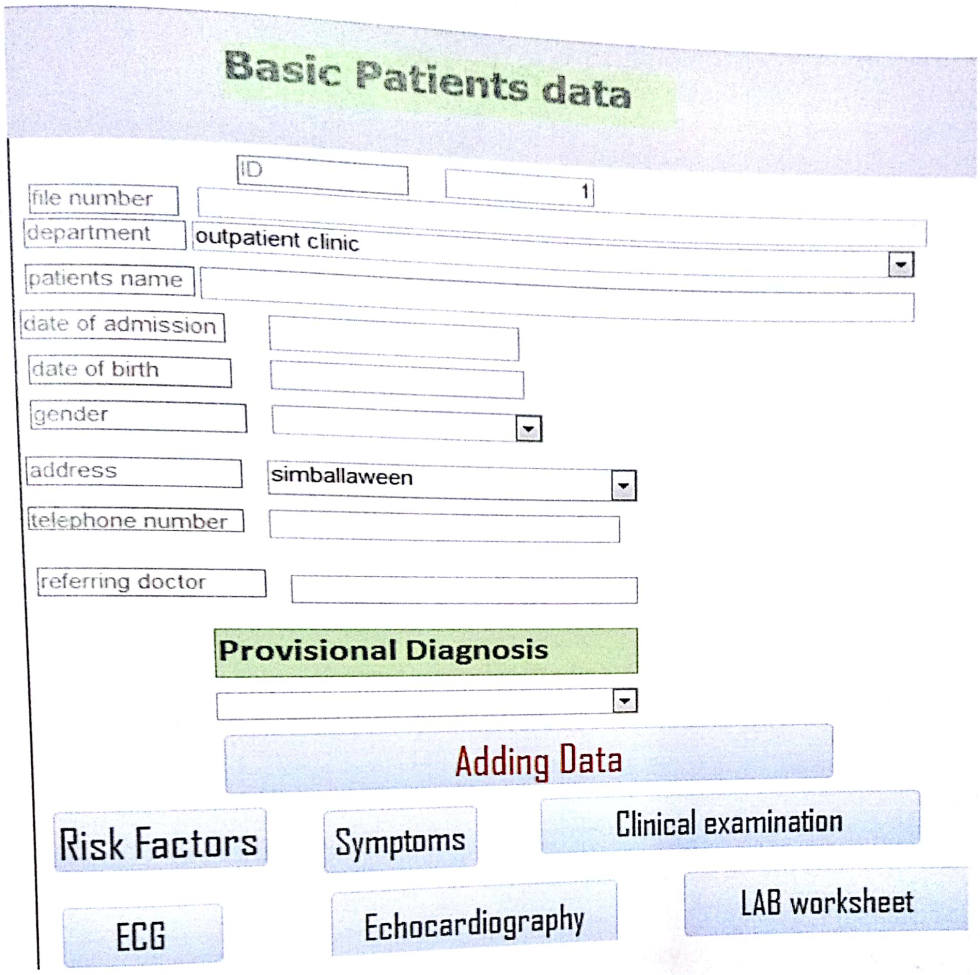


FIGURE 6: BASIC INFORMATION

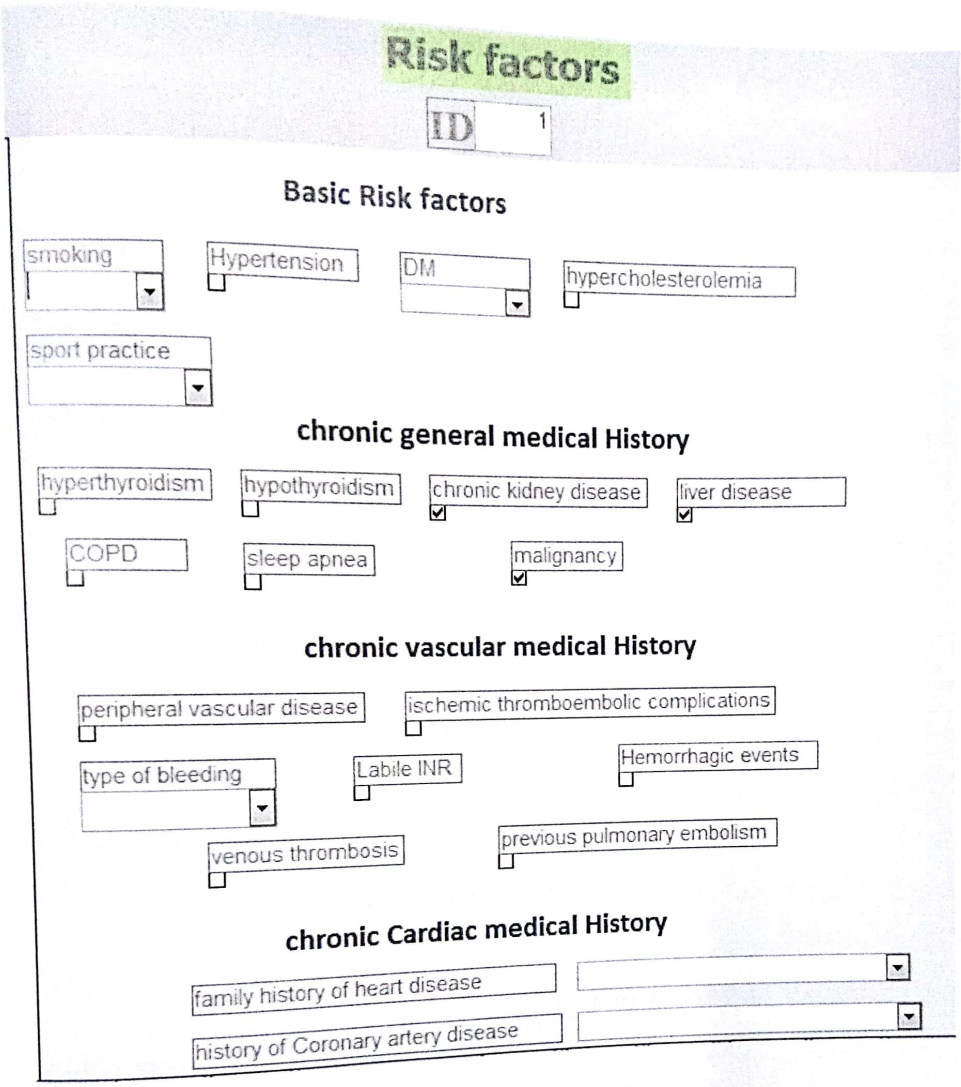


FIGURE 7: RISK FACTOR

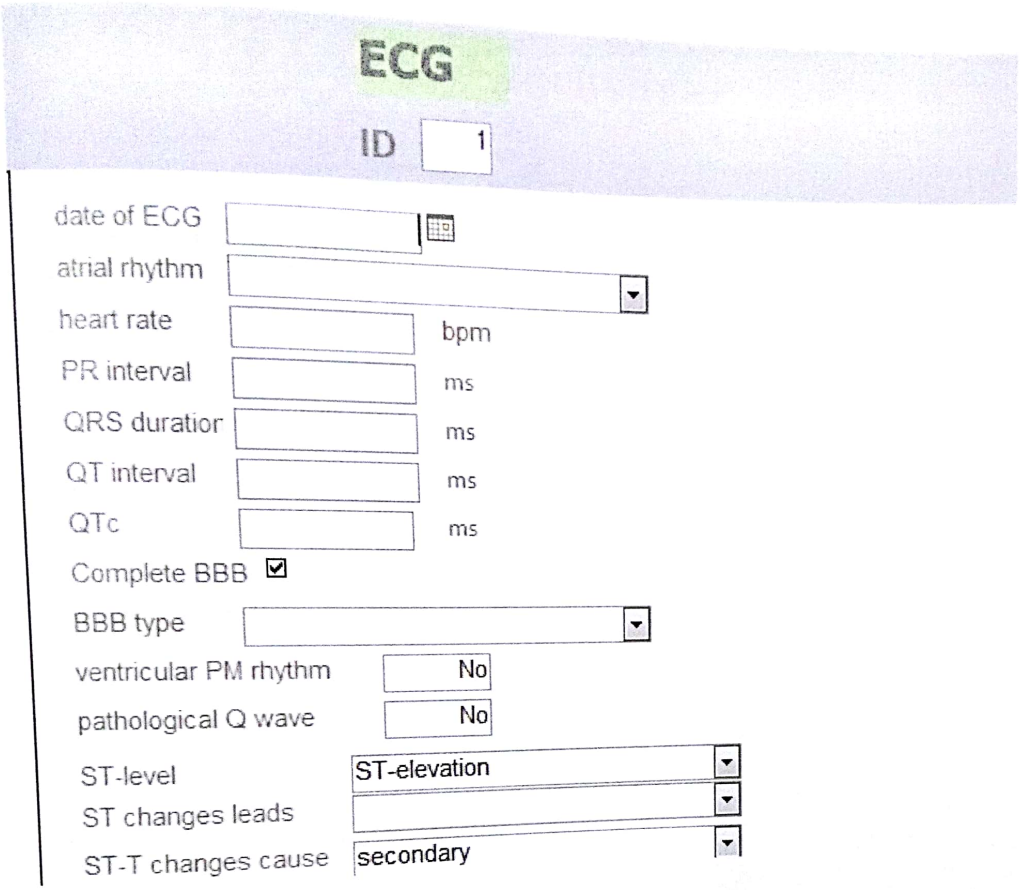

ECG
ID
1
date of ECG
atrial rhythm
heart rate
bpm
PR interval
ms
QRS duratior
ms
QT interval
ms
QTc
ms
Complete BBB
BBB type
ventricular PM rhythm
No
pathological Q wave
No
ST-level
ST-elevation
ST changes leads
ST-T changes cause
secondary

FIGURE 8: ECG

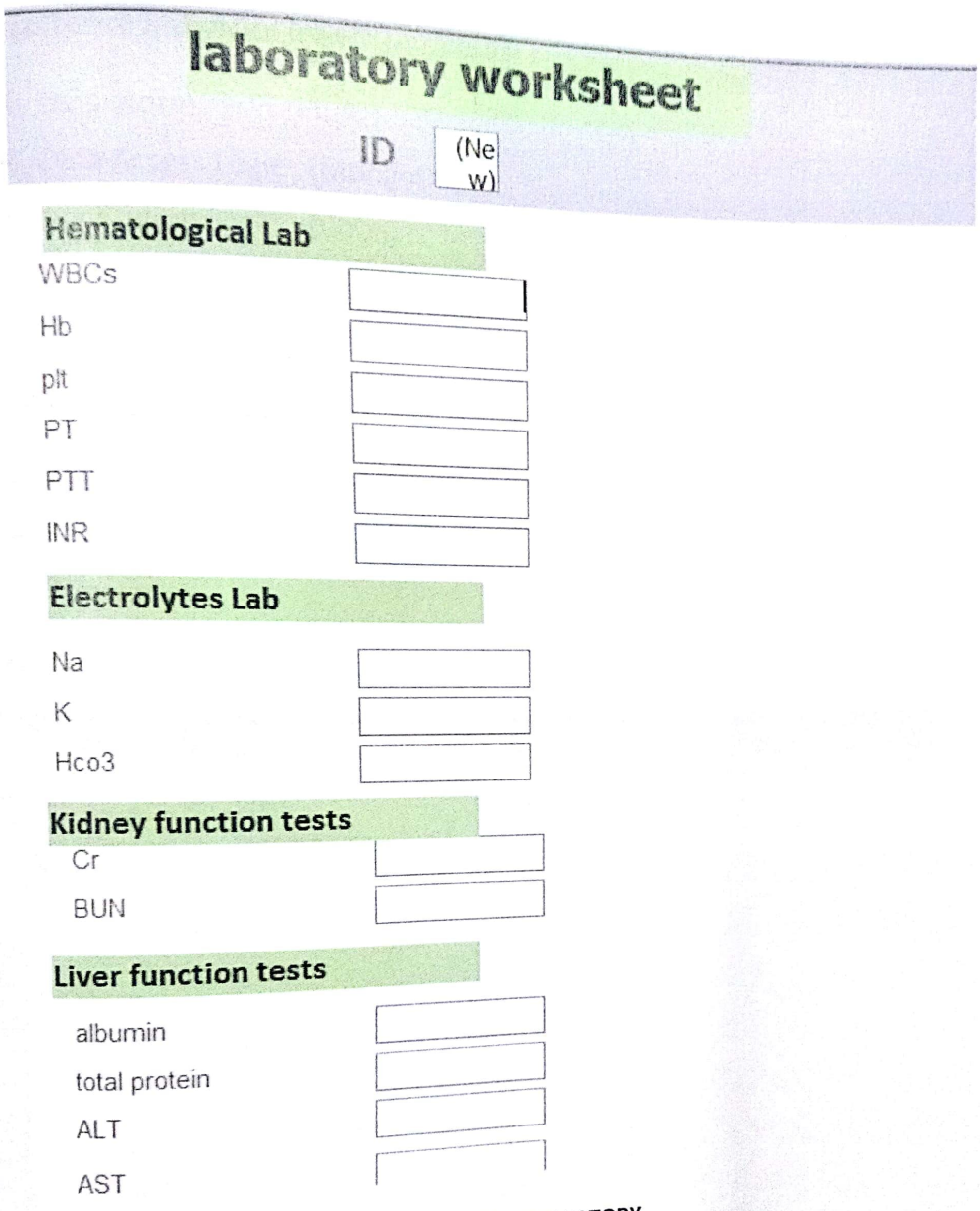


FIGURE 9: LABORATORY

ARCHITECTURE DESIGN

The objective of architecture design is to determine how software components of the information system will be assigned to the hardware device of the system.

The software is divided into

- Data Store
- Data Access Logic (MongoDB)
- Application Logic (Code)
- Presentation Logic (UI)

The Hardware id divided into

- Clients
- Servers
- Network

Servers:

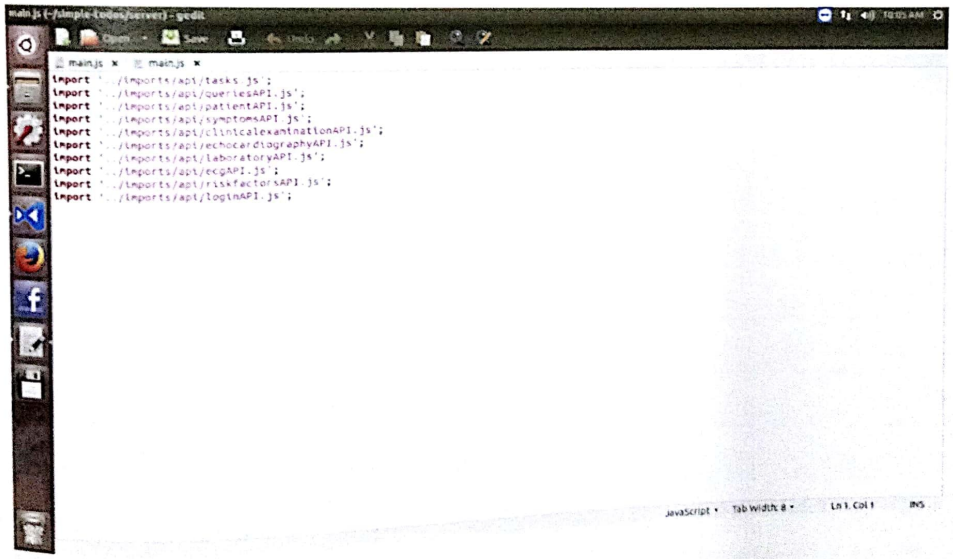


FIGURE 10:SERVER

Client:

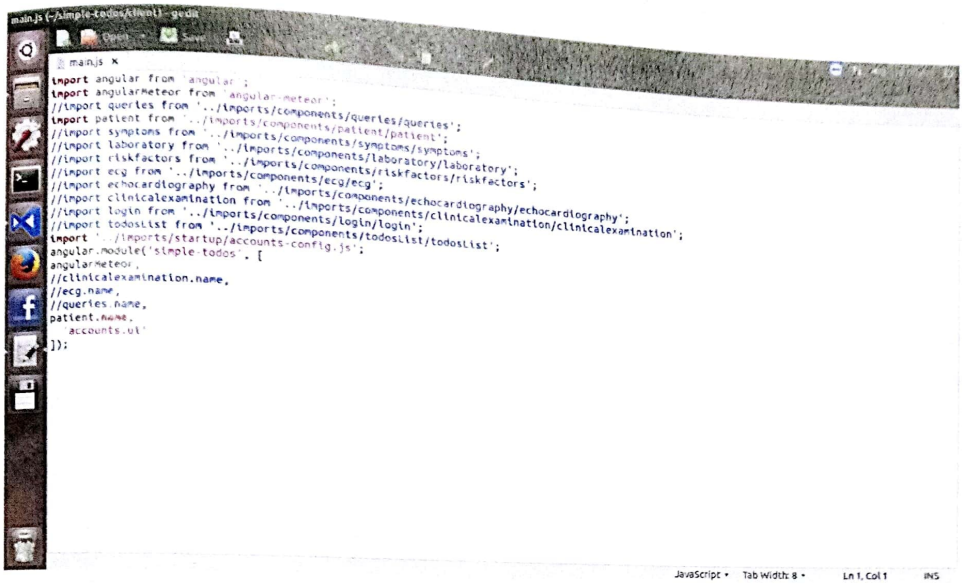


FIGURE 11:CLINET.JS

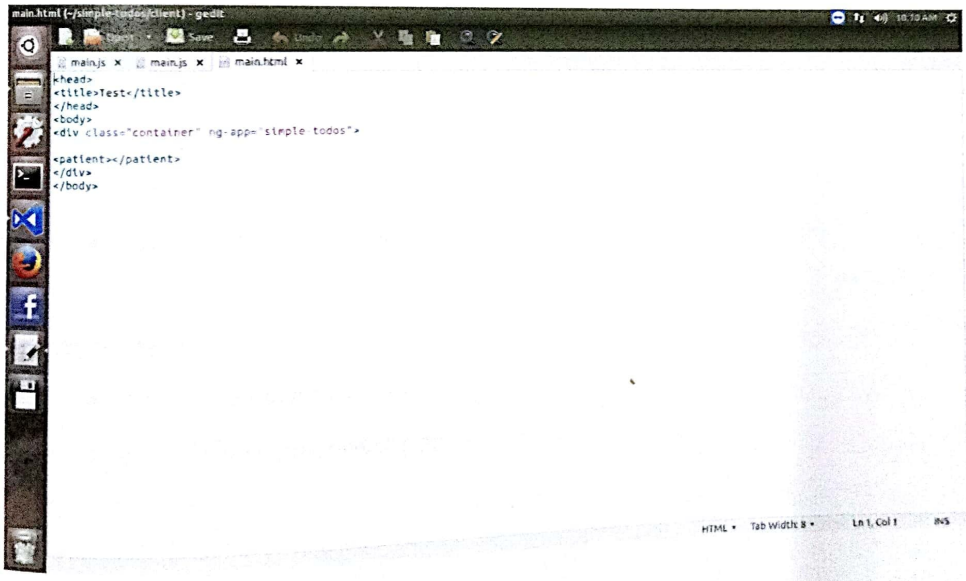


FIGURE 12:CLIENT.HTML

Client-Server Architectures

In these architectures, the client is responsible for the presentation logic, where the server is responsible for the data access logic and data storage. The application logic may reside on the client, reside on the server, or by split between both. If the client contain all or most of the application logic, it is called thick or fat client.

NAVIGATION DESIGN

The navigation component of the interface enables the user to enter commands to navigate through the system and perform actions to enter and review information it contains.

- **Basic principles:**
  - Prevent mistakes.
  - Simplify recovery from mistakes.
  - Use consistent.

- **Types of navigation control:**
  - Language.
  - Menus.
  - Direct manipulation.

- **Messages:**
  - Error messages.
  - Confirmation messages.

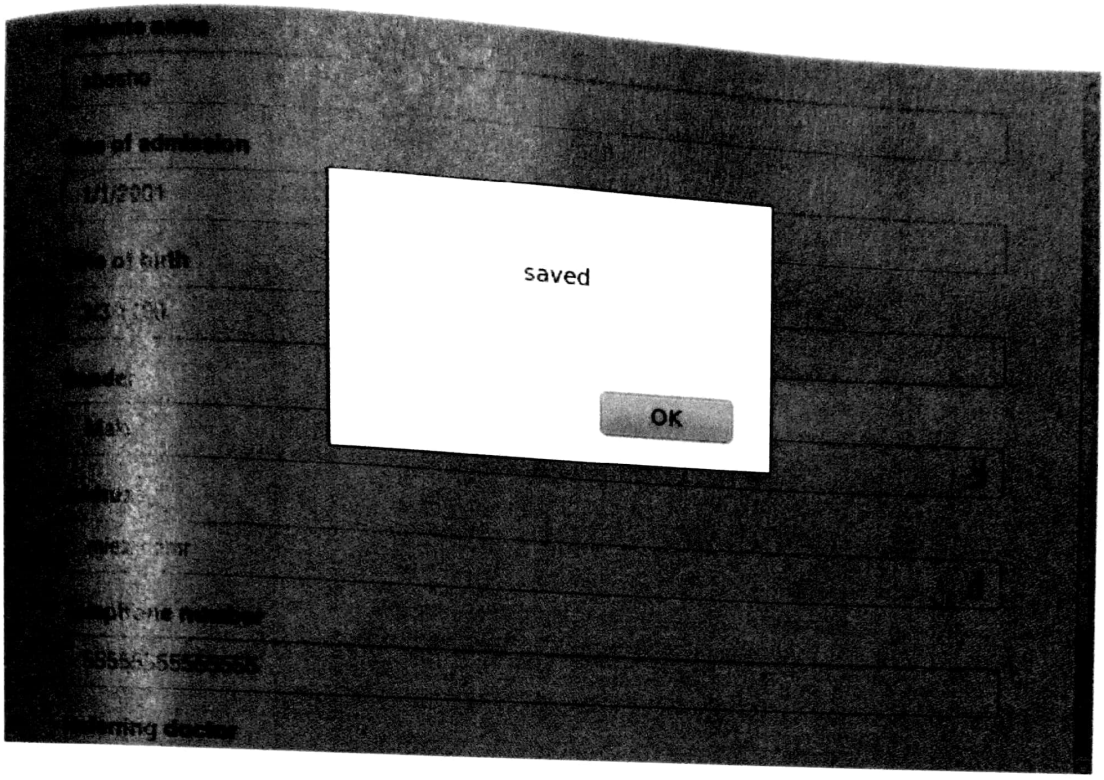


FIGURE 13:CONFIRMATION MESSAGE

INPUT DESIGN

The goal of input design is to simply and easily capture accurate information for the system, typically by using online or batch processing, capturing data at the source, and minimizing keystrokes. Input design includes both the design of input screens and all preprinted forms that are used to collect data before they are entered into the information system.

Types of inputs

- Text.
- Numbers.
- Selection Box.
- Radio Box

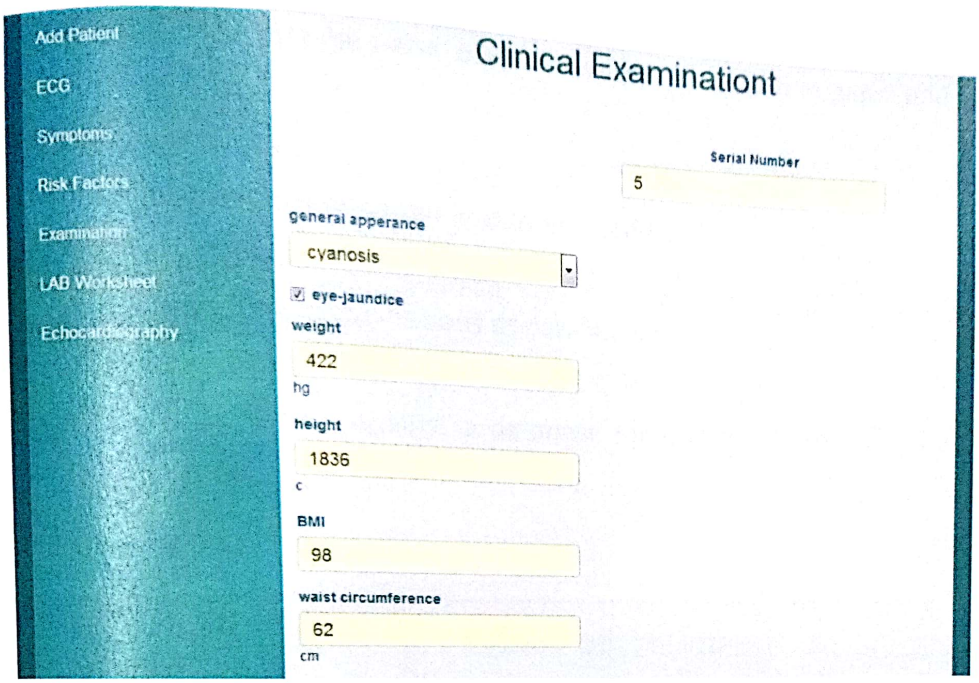


FIGURE 14:CLINICAL EXAMINATION

INPUT VALIDATION

All data entered into the system must be validated in order to ensure accuracy. Input validation (also called edit checks) can take many forms. Ideally, to prevent invalid information from entering the system, computer systems should not accept data that fail any important validation check.

- Completeness check
  - Ensures that all required data have been entered.
- Format check
  - Ensures that data are of the right type (e.g., numeric) and in the right format
  - (e.g., month, day, year).
- Range check

- Ensures that numeric data are within correct minimum and max values.

- Check digit check
  - Check digits are added to numeric codes.
- Consistency checks
  - Ensure that combinations of data are valid.
- Database checks
  - Compare data against a database (or file) to ensure that they are correct.

Examples of Inputs:

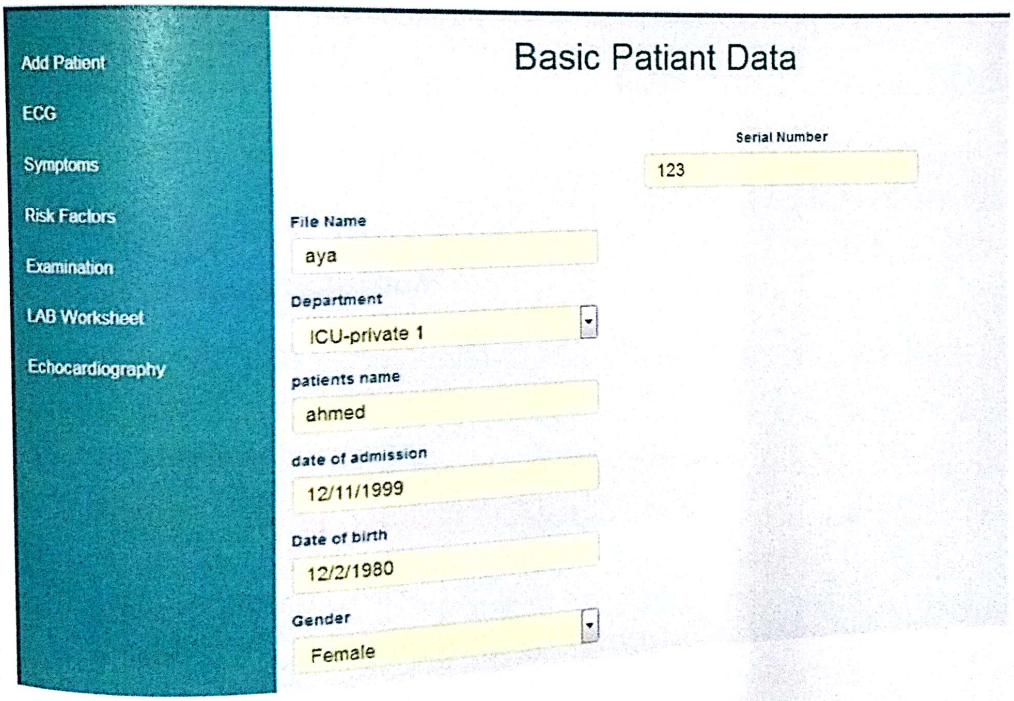


FIGURE 15:ADD PATIENT

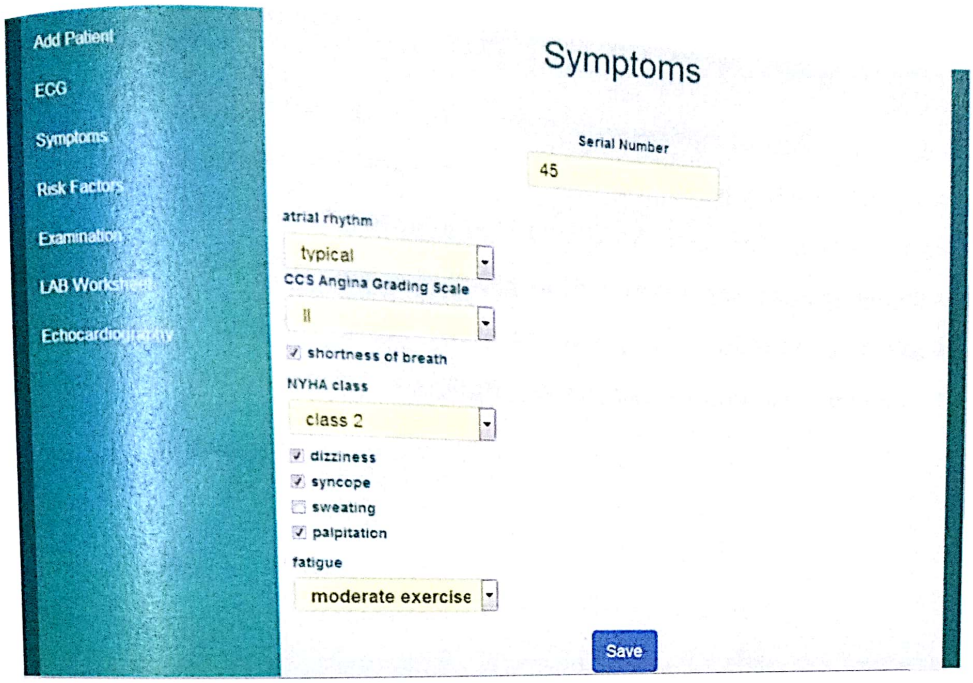


FIGURE 16:SYMPTOMS

Examples of inputs validation:

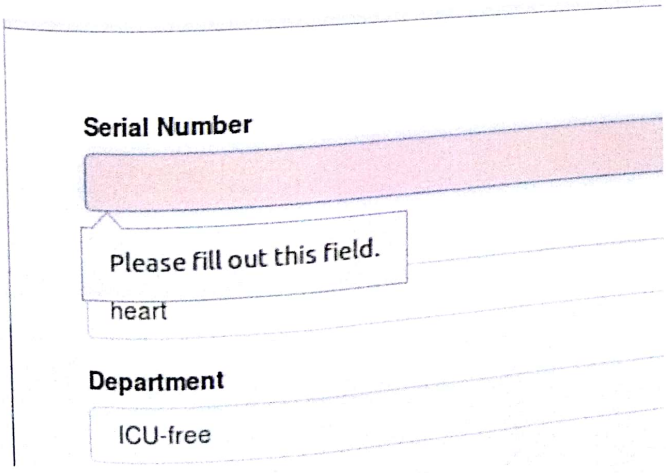


FIGURE 17:INPUT VALIDATION

INPUT DESIGN

The goal of output design is to present information to users so that they can accurately understand it with the least effort, usually by understanding how reports will be used and designing them to minimize information overload and bias. Output design means designing both screens and reports in other media, such as paper and the Web. There are many types of reports, including detail reports, summary reports, exception reports, turnaround documents, and graph

SYSTEM IMPLEMENTATION

IMPLEMENTATION PHASE

The final phase in the SDLC is the implementation phase, during which the system is actually built (or purchased, in the case of a packaged software design). At the end of implementation, the final system is put into operation and supported and maintained.

Objective

- Moving into Implementation
- Transition to the New System

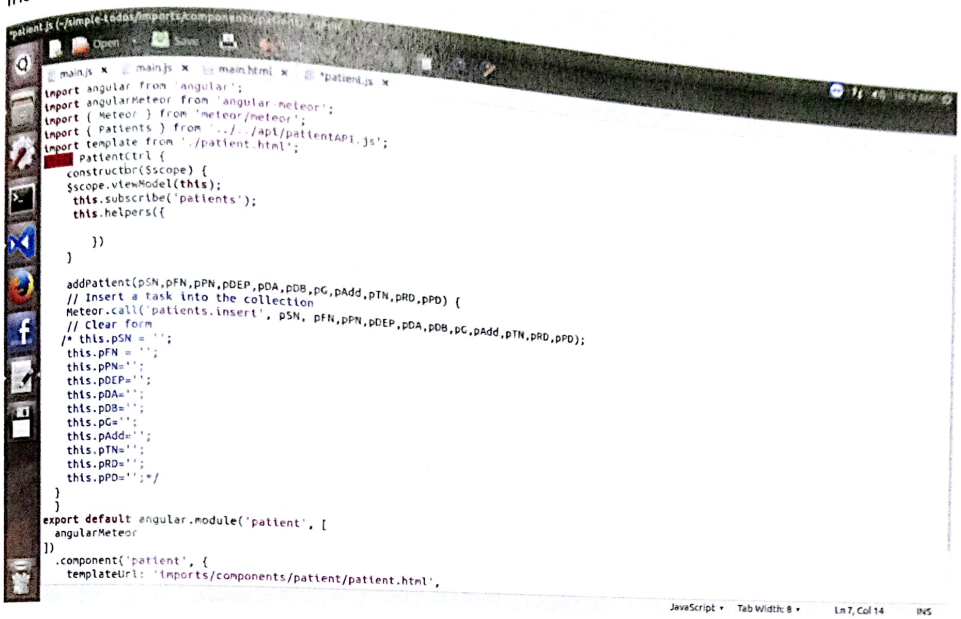
By now you can realize that we have four main operations that can be carried out

- Update
- Insert
- Delete
- Check

We will be showing you some examples of code for each of these operations

But first let's have a look at server connection code

Insert Code:

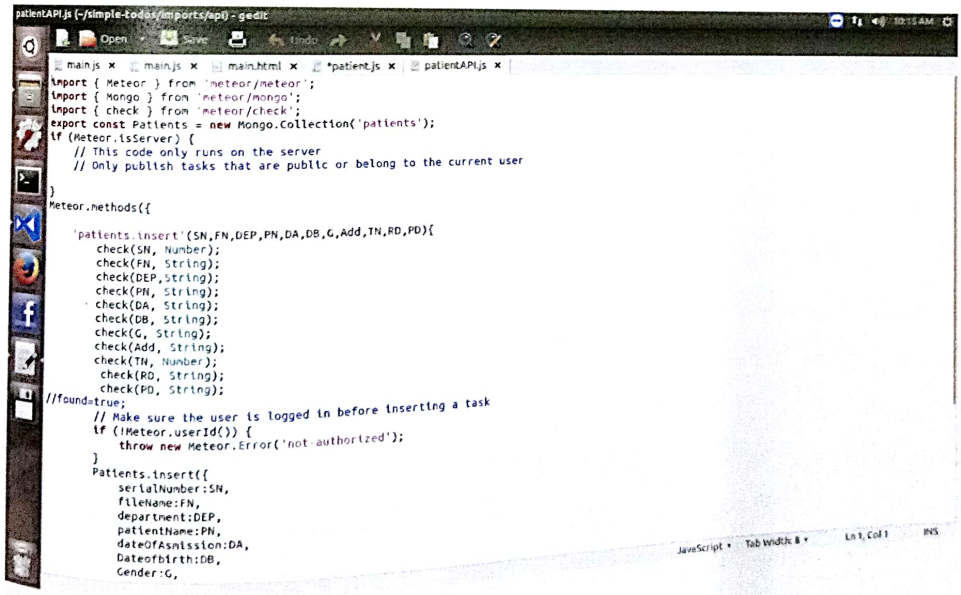


```
import angular from 'angular';
import angularMeteor from 'angular-meteor';
import { Meteor } from 'meteor/meteor';
import { Patients } from '../../api/patientAPI.js';
import template from './patient.html';
class PatientCtrl {
  constructor($scope) {
  $scope.viewModel(this);
   this.subscribe('patients');
   this.helpers({

      })
  }

  addPatient(pSN,pFN,pPN,pDEP,pDA,pDB,pG,pAdd,pTN,pRD,pPD) {
  // Insert a task into the collection
  Meteor.call('patients.insert', pSN, pFN,pPN,pDEP,pDA,pDB,pG,pAdd,pTN,pRD,pPD);
  // Clear form
 /* this.pSN = '';
  this.pFN = '';
  this.pPN='';
  this.pDEP='';
  this.pDA='';
  this.pDB='';
  this.pG='';
  this.pAdd='';
  this.pTN='';
  this.pRD='';
  this.pPD='';*/
 }
 }
export default angular.module('patient', [
  angularMeteor
])
  .component('patient', {
    templateUrl: 'imports/components/patient/patient.html',
```

FIGURE 18:INSERT CODE



```
import { Meteor } from 'meteor/meteor';
import { Mongo } from 'meteor/mongo';
import { check } from 'meteor/check';
export const Patients = new Mongo.Collection('patients');
if (Meteor.isServer) {
   // This code only runs on the server
   // Only publish tasks that are public or belong to the current user

}
Meteor.methods({

   'patients.insert'(SN,FN,DEP,PN,DA,DB,G,Add,TN,RD,PD){
      check(SN, Number);
      check(FN, String);
      check(DEP,String);
      check(PN, String);
      check(DA, String);
      check(DB, String);
      check(G, String);
      check(Add, String);
      check(TN, Number);
       check(RD, String);
       check(PD, String);
//found=true;
      // Make sure the user is logged in before inserting a task
      if (!Meteor.userId()) {
         throw new Meteor.Error('not-authorized');
      }
      Patients.insert({
         serialNumber:SN,
         fileName:FN,
         department:DEP,
         patientName:PN,
         dateOfAsmission:DA,
         Dateofbirth:DB,
         Gender:G,
```

FIGURE 19:INSERT CODE

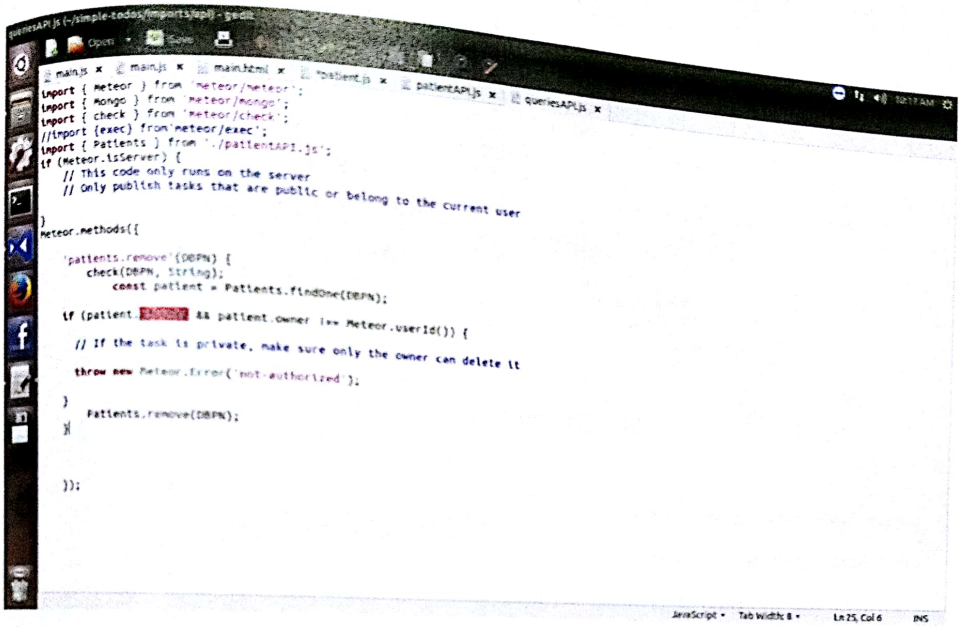


FIGURE 20:DELETE CODE

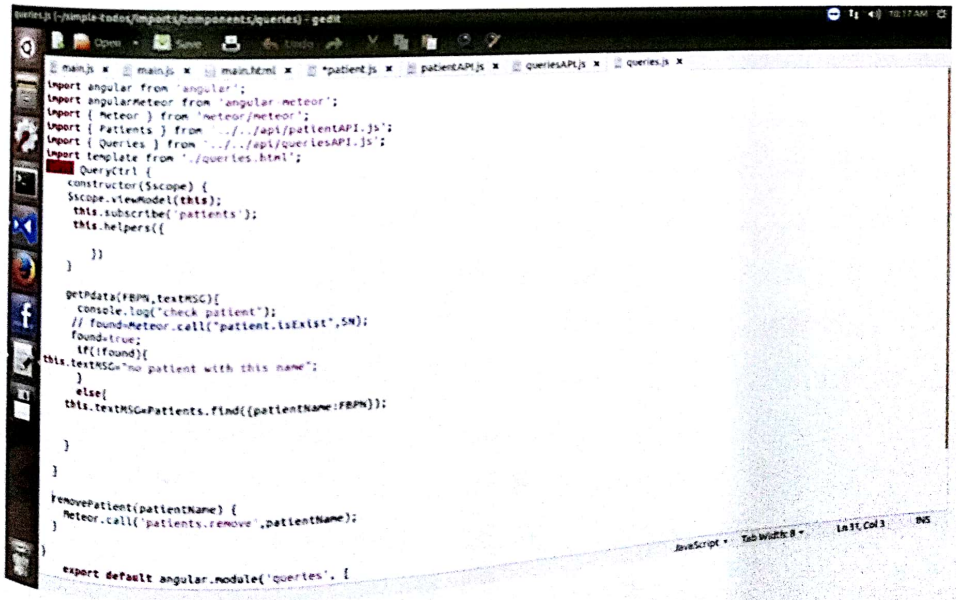


FIGURE 21:CHECK CODE

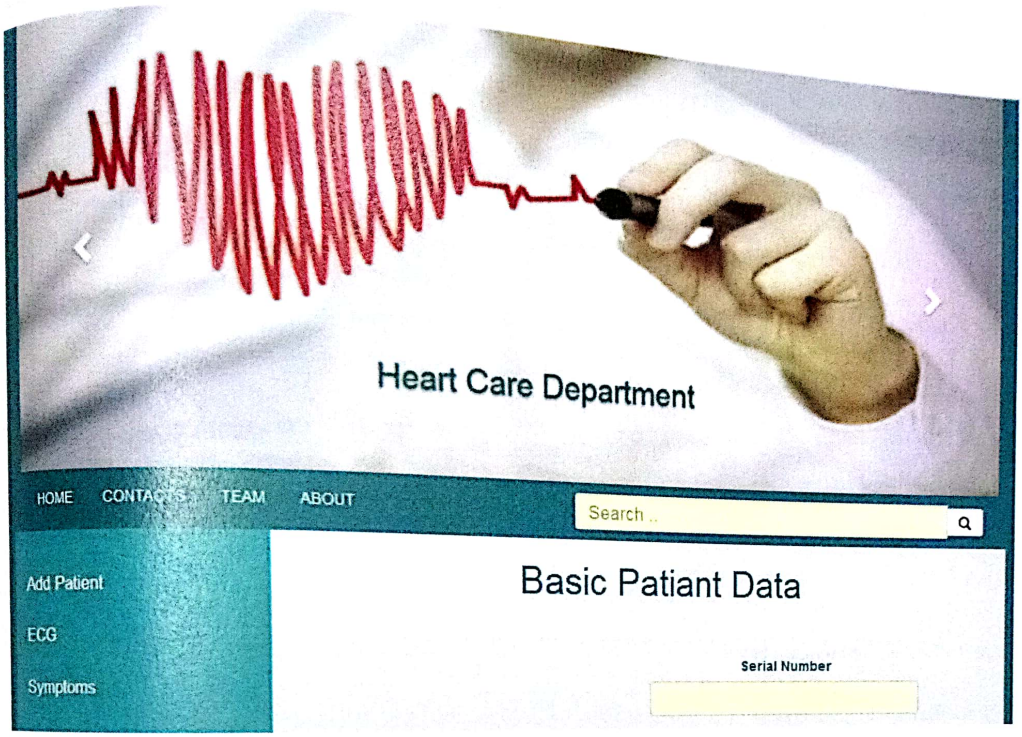
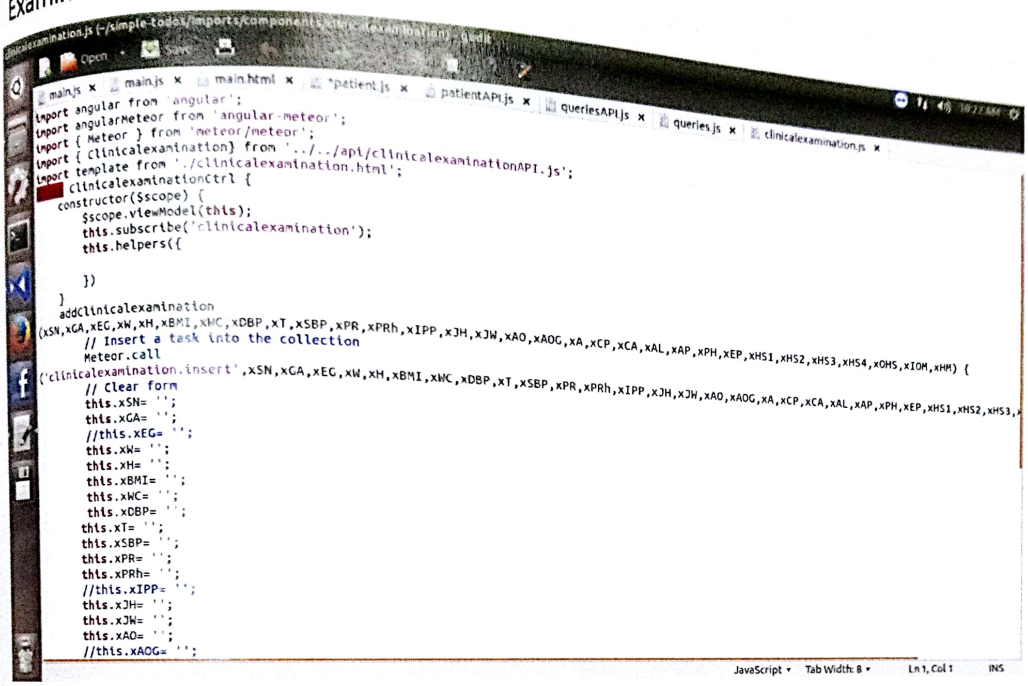


FIGURE 22:HPME SCREEN

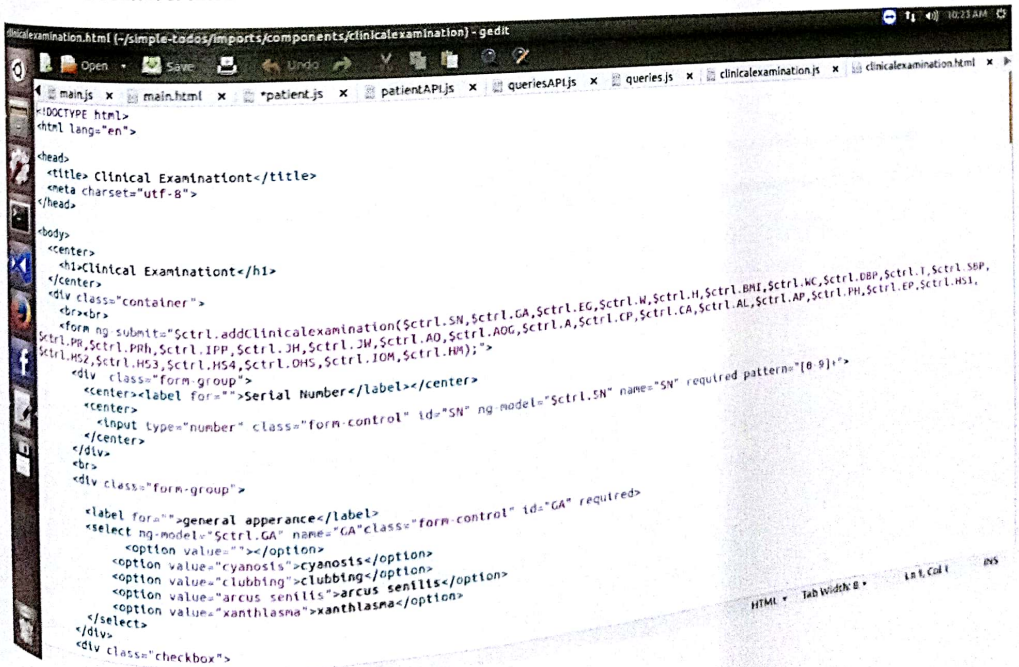
Examination code:



```
import angular from 'angular';
import angularMeteor from 'angular-meteor';
import { Meteor } from 'meteor/meteor';
import { clinicalexamination} from '../../api/clinicalexaminationAPI.js';
import template from './clinicalexamination.html';
class clinicalexaminationCtrl {
  constructor($scope) {
    $scope.viewModel(this);
    this.subscribe('clinicalexamination');
    this.helpers({

    })
  }
  addclinicalexamination
(xSN,xGA,xEG,xW,xH,xBMI,xWC,xDBP,xT,xSBP,xPR,xPRh,xIPP,xJH,xJW,xAO,xAOG,xA,xCP,xCA,xAL,xAP,xPH,xEP,xHS1,xHS2,xHS3,xHS4,xOHS,xIOM,xHM) {
    // Insert a task into the collection
    Meteor.call
('clinicalexamination.insert',xSN,xCA,xEG,xW,xH,xBMI,xWC,xDBP,xT,xSBP,xPR,xPRh,xIPP,xJH,xJW,xAO,xAOG,xA,xCP,xCA,xAL,xAP,xPH,xEP,xHS1,xHS2,xHS3,x
    // Clear form
    this.xSN= '';
    this.xGA= '';
    //this.xEG= '';
    this.xW= '';
    this.xH= '';
    this.xBMI= '';
    this.xWC= '';
    this.xDBP= '';
   this.xT= '';
   this.xSBP= '';
   this.xPR= '';
   this.xPRh= '';
   //this.xIPP= '';
   this.xJH= '';
   this.xJW= '';
   this.xAO= '';
   //this.xAOG= '';
```

FIGURE 23:CLINICAL.JS



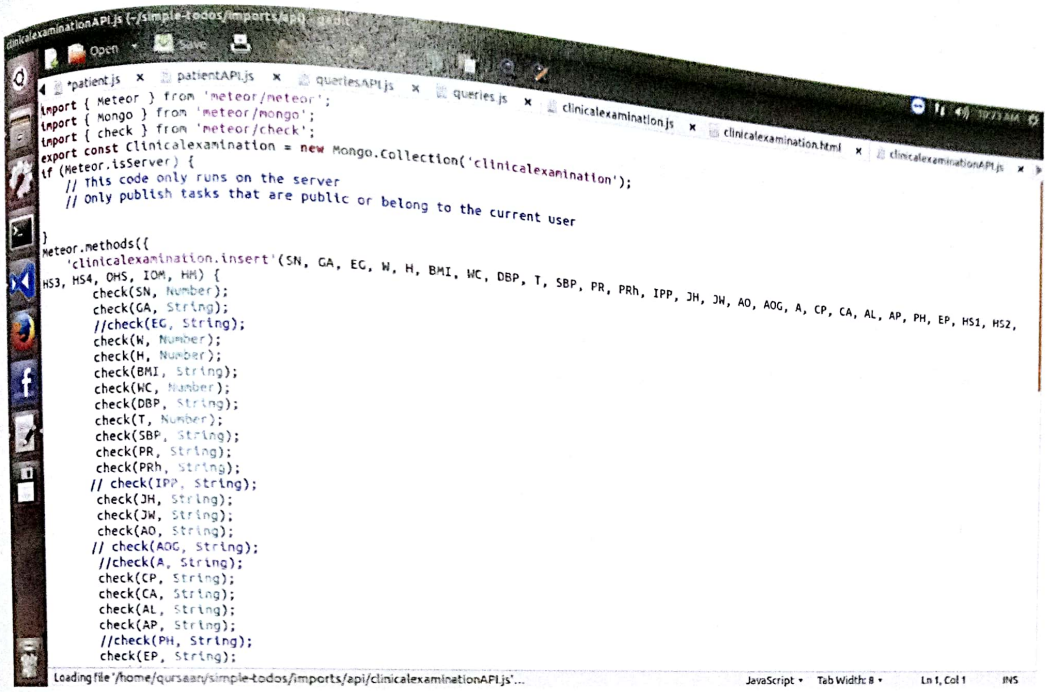
```
<!DOCTYPE html>
<html lang="en">

<head>
  <title> Clinical Examinationt</title>
  <meta charset="utf-8">
</head>

<body>
  <center>
    <h1>Clinical Examinationt</h1>
  </center>
  <div class="container">
    <br><br>
    <form ng-submit="$ctrl.addClinicalexamination($ctrl.SN,$ctrl.GA,$ctrl.EG,$ctrl.W,$ctrl.H,$ctrl.BMI,$ctrl.WC,$ctrl.DBP,$ctrl.T,$ctrl.SBP,
$ctrl.PR,$ctrl.PRh,$ctrl.IPP,$ctrl.JH,$ctrl.JW,$ctrl.AO,$ctrl.AOG,$ctrl.A,$ctrl.CP,$ctrl.CA,$ctrl.AL,$ctrl.AP,$ctrl.PH,$ctrl.EP,$ctrl.HS1,
$ctrl.HS2,$ctrl.HS3,$ctrl.HS4,$ctrl.OHS,$ctrl.IOM,$ctrl.HM);">
      <div class="form-group">
        <center><label for="">Serial Number</label></center>
        <center>
          <input type="number" class="form-control" id="SN" ng-model="$ctrl.SN" name="SN" required pattern="[0-9]+">
        </center>
      </div>
      <br>
      <div class="form-group">

        <label for="">general apperance</label>
        <select ng-model="$ctrl.GA" name="GA"class="form-control" id="GA" required>
            <option value=""></option>
            <option value="cyanosis">cyanosis</option>
            <option value="clubbing">clubbing</option>
            <option value="arcus senilis">arcus senilis</option>
            <option value="xanthlasma">xanthlasma</option>
        </select>
      </div>
      <div class="checkbox">
```

FIGURE 24:CLINICAL.HTML



```
import { Meteor } from 'meteor/meteor';
import { Mongo } from 'meteor/mongo';
import { check } from 'meteor/check';
export const Clinicalexamination = new Mongo.Collection('clinicalexamination');
if (Meteor.isServer) {
  // This code only runs on the server
  // Only publish tasks that are public or belong to the current user

}
Meteor.methods({
  'clinicalexamination.insert'(SN, GA, EG, W, H, BMI, WC, DBP, T, SBP, PR, PRh, IPP, JH, JW, AO, AOG, A, CP, CA, AL, AP, PH, EP, HS1, HS2,
HS3, HS4, OHS, IOM, HM) {
      check(SN, Number);
      check(GA, String);
      //check(EG, String);
      check(W, Number);
      check(H, Number);
      check(BMI, String);
      check(WC, Number);
      check(DBP, String);
      check(T, Number);
      check(SBP, String);
      check(PR, String);
      check(PRh, String);
     // check(IPP, String);
      check(JH, String);
      check(JW, String);
      check(AO, String);
     // check(AOG, String);
      //check(A, String);
      check(CP, String);
      check(CA, String);
      check(AL, String);
      check(AP, String);
      //check(PH, String);
      check(EP, String);
```

FIGURE 25:CLINICALAPI.JS

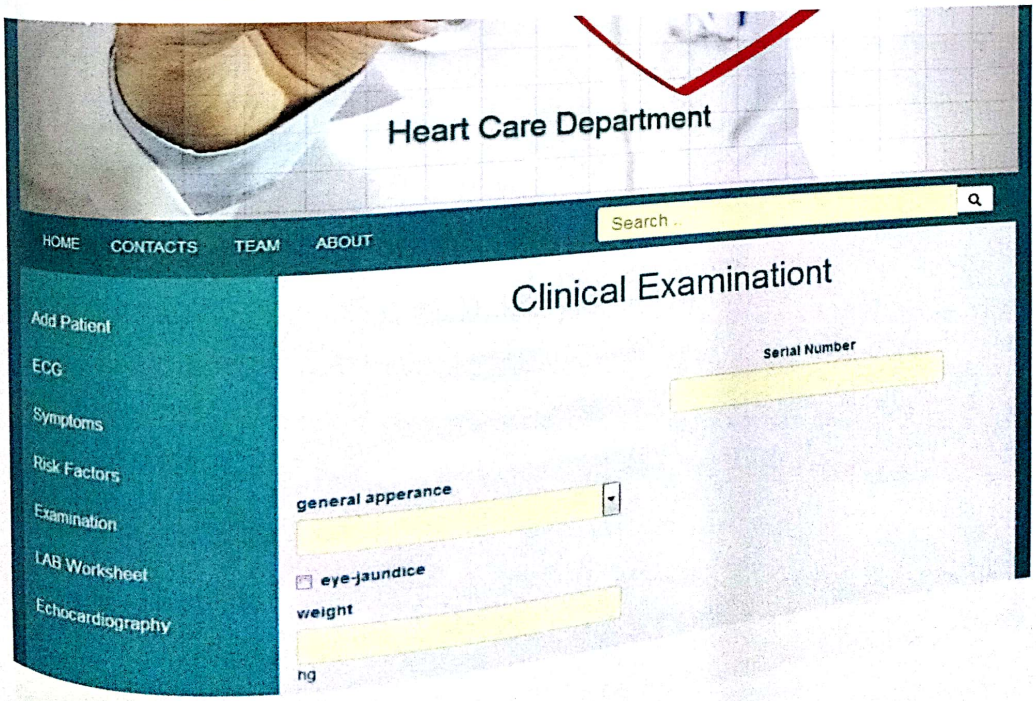


FIGURE 26:CLINICAL EXAMINATION INTERFACE

Contact page:

```
<body>
  <form class="sub-form">
  <div class="input-contain">
    <div class="circle circle-quill">
      <svg class="quill" x="0px" y="0px"
      width="25.167px" height="25.167px" viewBox="0 0 25.167 25.167" enable-background
      ="new 0 0 25.167 25.167" xml:space="preserve">
      <path fill="#4e3332" d="M0,25C3.125,15.625,11.304,0,25,0c-6.421,5.151-9.375,12.1
      88-14.062,17.188s-4.688,0-4.688,0L1.562,25H0z"/>
      </svg>
    </div>
    <div class="circle circle-paper">
      <svg class="paper" x="0" y="0" width="25.1" height="25.1" viewBox="0 0 25.1 25.1
      " enable-background="new 0 0 25.125 25.125" xml:space="preserve"><path fill="
      #b1dbd3" d="M24 2.1C23.5 2.3 1.2 10.2 0.8 10.3c-0.4 0.1-0.5 0.5 0 0.6 0.5 0.2 5
      2 5 2H5.813 1.2c0 0 14.2-10.4 14.4-10.6 0.2-0.1 0.4 0.1 0.3 0.3 -0.1 0.2-10.3
      11.2-10.3 11.2 0 0 0 0 0 0l-0.6 0.7 0.8 0.4c0 0 6.1 3.3 6.5 3.5 0.4 0.2 0.9 0
      1-0.4 0.1-0.6 3.7-16.1 3.8-16.4C24.7 2.3 24.4 2 24 2.1zM8.7 21.2c0 0.3 0.2 0.4
      0.4 0.2 0.3-0.3 3.7-3.4 3.7-3.4l-4.2-2.2V21.2z"/></svg>
    </div>
    <h2 class="info">want more information?</h2>
    <h2 class="success">Success!</h2>
    <input type="text" id="firstname" name="firstname" required="true" placeholder="
    full name" required maxlength="100" />
    <input type="email" placeholder="email" name="email" required="required" />

    <input type="phone" placeholder="phone" name="phone" required="required" />

    <input type="message" placeholder="message" name="message"required="true" required
     maxlength="auto" />
    <div class="allsub">
      <div class="submit">Contact Us Today</div>
      <div class="submit-under"></div>
    </div><!--allsub-->
  <svg class="loader" x="0px" y="0px" width="55px" height="55px" viewBox="0 0 55 55"
```

FIGURE 27:CONTACT CODE

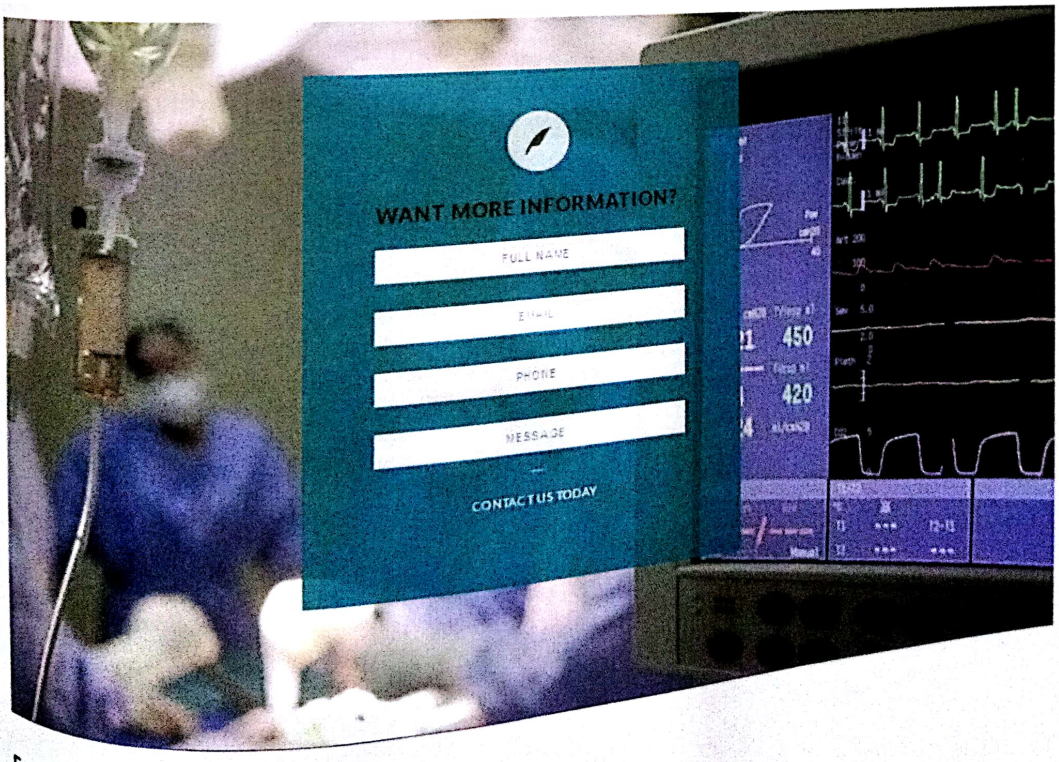


FIGURE 28:CONTACT INTERFACE

TESTING AND EVALUATION

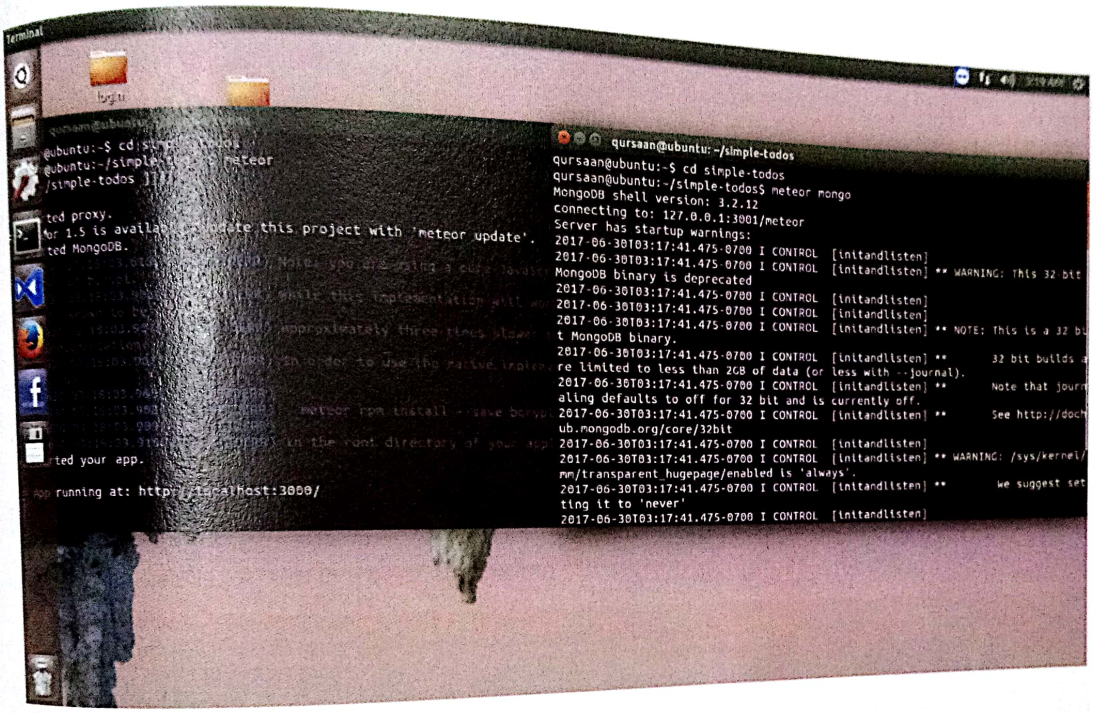


FIGURE 29:RUNNING APP

PATIENT INFORMATION

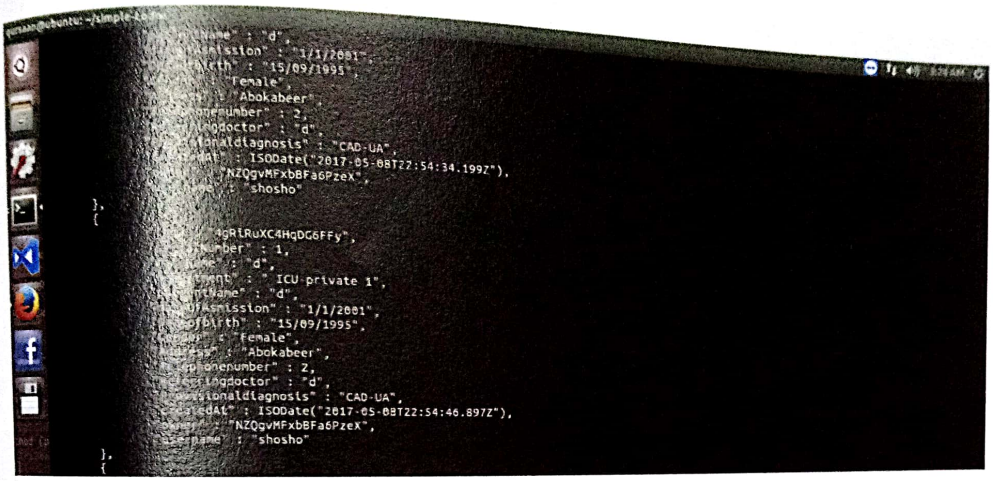


FIGURE 30:SHOW PATIENTS IN THE DATABASE

FIGURE 31:SHOW SPECIFIC PATIENT INFORMATION

After removing a patient when we test:

FIGURE 32:PATIENT NOT FOUND

Another Example of documents in DB:

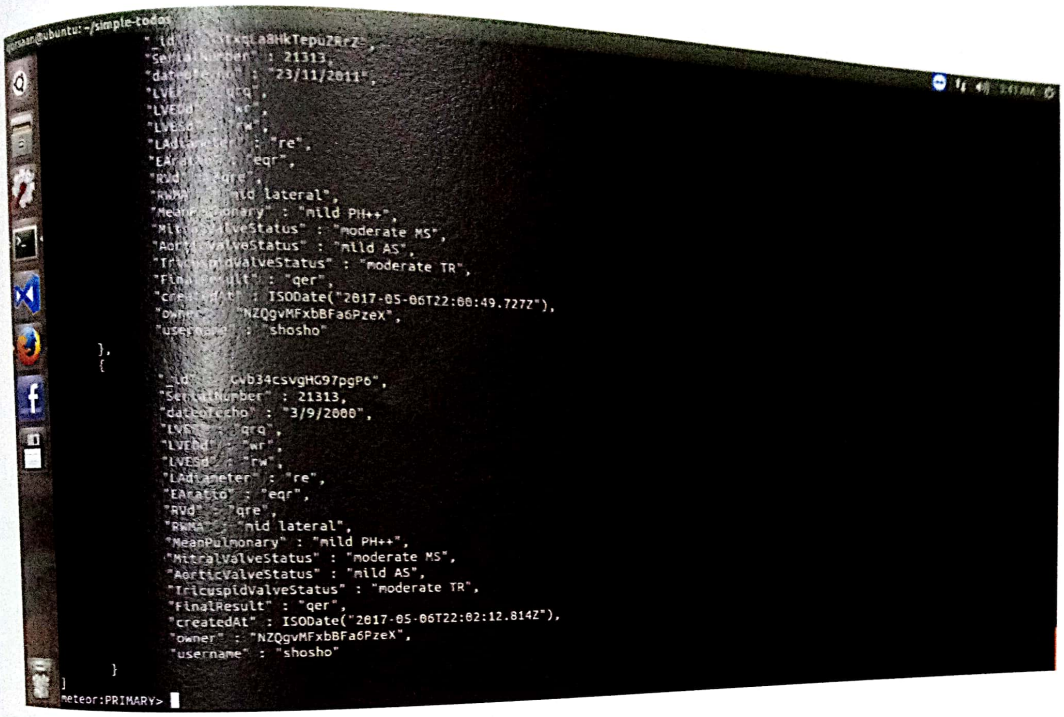


FIGURE 33:INFORMATION OF ECG

FUTURE WORK

In the future we plan to use this system to conduct user studies. We hope to determine its usefulness to documenters and to what degree this system can improve the quality of information located by users. We also hope to more fully integrate the selection of hypertext terms into the system.

The current method is effective, but could become cumber-some. We also plan to investigate the system's usefulness as a method to track documents produced during different phases of the software life cycle. This methodology could be applied to organize and improve access to those documents. As mentioned earlier, we are also considering an Index to be used with the hardcopy version of the documentation. A custom filter similar to those provided for hyper-text links and cross-references should be provided to facilitate the comprehensive indexing of chosen terms

REFERENCES

1-*Meteor*: Build Apps with JavaScript

https://www.meteor.com/

2-*Angular Tutorial*: Build a simple app to manage a task list.

https://www.meteor.com/tutorials/angular/creating-an-app

3- *Introducing Meteor API Docs*

http://docs.meteor.com/#/full/

4-*MongoDB Tutorial*: learn mongo DB nosql document database

https://www.tutorialspoint.com/mongodb/

5-*Doc*: Node.js

https://nodejs.org/en/docs/

6- *Why Meteor*

http://whymeteor.com/

7-*Angular JS*

https://en.wikipedia.org/wiki/AngularJS

https://angularjs.org/

8-*Query document*

https://docs.mongodb.com/manual/tutorial/query-documents/

9-*Chapter 8 Validation*

http://meteortips.com/second-meteor-tutorial/validation/

10-URLS & Routings

https://guide.meteor.com/routing.html

11-INTERMEDIATE METEOR
https://www.youtube.com/watch?v=BI8IslJHSag&list=PLLnpHn493BHFYZUSK62a
VycgcAouqBt7V

APPENDIX A

Meteor: a full-stack JavaScript platform for developing modern web and mobile applications.

Framework: an essential supporting structure of a building, vehicle, or object.

"a conservatory in a delicate framework of iron".

JavaScript: is a high-level, dynamic, untipped, and interpreted programming language.

MongoDB: is a free and open-source cross-platform document-oriented database program.

Document: is a set of key-value pairs.