**Logo, company name

Description automatically generated**

**COMSATS University Islamabad (CUI)**

Software Design Description  
(SDS DOCUMENT)

for

**Spatial Neglect**

Version 1.0

***By***

**Danish Ali CUI/FA18-BCS-017/ISB**

**Malaika Atiq CUI/FA18-BCS-040/ISB**

***Supervisor*Mr. Khurram Iqbal**

*Bachelor of Science in Computer Science (2018-2022)*

**Table of Contents**

Contents

[1. Introduction 1](#_Toc89640393)

[2. Design Methodology and Software Process Model 1](#_Toc89640394)

[3. System Overview 1](#_Toc89640395)

[3.1 Architectural Design 2](#_Toc89640396)

[3.1.1 Box and Line Diagram: 2](#_Toc89640397)

[3.1.2 Architecture Diagram: 2](#_Toc89640398)

[3.1.3 MVC Components: 3](#_Toc89640399)

[4. Design Models 4](#_Toc89640400)

[4.1 Activity Diagrams: 4](#_Toc89640401)

[4.1.1 Activity Diagram for Detector: 4](#_Toc89640402)

[4.1.2 Activity Diagram for Schedule Appointment: 5](#_Toc89640403)

[4.1.3 Activity Diagram for Review: 6](#_Toc89640404)

[4.1.4 Activity Diagram for Rehabilitator: 7](#_Toc89640405)

[4.1.5 Activity Diagram for Post request: 8](#_Toc89640406)

[4.1.6 Activity Diagram for Respond to Request: 9](#_Toc89640407)

[4.1.7 Activity Diagram for Payments: 10](#_Toc89640408)

[4.1.8 Activity Diagram for Report Generator: 11](#_Toc89640409)

[4.2 Sequence Diagrams: 12](#_Toc89640410)

[4.2.1 Sequence Diagram for Scheduling Appointments: 12](#_Toc89640411)

[4.2.2 Sequence Diagram for Detector: 13](#_Toc89640412)

[4.2.3 Sequence Diagram for Rehabilitator: 14](#_Toc89640413)

[4.2.4 Sequence Diagram for Patient Logging: 15](#_Toc89640414)

[4.2.5 Sequence Diagram for Report Generator: 16](#_Toc89640415)

[4.3 State Transition Diagrams: 16](#_Toc89640416)

[4.3.1 State transition Diagram for Rehabilitator: 17](#_Toc89640417)

[4.3.2 State transition Diagram for Detector: 17](#_Toc89640418)

[4.3.3 State transition Diagram for Report Generator: 18](#_Toc89640419)

[5. Data Design 20](#_Toc89640420)

[5.1 Major Collections: 21](#_Toc89640421)

[5.2 Data Structures: 21](#_Toc89640422)

[5.2.1 Data Structure for Guardian: 21](#_Toc89640423)

[5.2.2 Data Structure for Reviews: 22](#_Toc89640424)

[5.2.3 Data Structure for Review Reply: 23](#_Toc89640425)

[5.2.4 Data Structure for Appointment: 23](#_Toc89640426)

[5.2.5 Data Structure for Statistics: 24](#_Toc89640427)

[5.2.6 Data Structure for FAQ: 24](#_Toc89640428)

[5.2.7 Data Structure for Doctor: 25](#_Toc89640429)

[5.2.8 Data Structure for Patient: 26](#_Toc89640430)

[5.2.9 Data Structure for Admin: 27](#_Toc89640431)

[5.2.10 Data Structure for Payments Plan: 28](#_Toc89640432)

[5.2.11 Data Structure for Payments: 29](#_Toc89640433)

[5.3 Data Dictionary 29](#_Toc89640434)

[5.3.1 Data Dictionary for Guardian Account 30](#_Toc89640435)

[5.3.2 Data Dictionary for Reviews 30](#_Toc89640436)

[5.3.3 Data Dictionary for Reviews reply 30](#_Toc89640437)

[5.3.4 Data Dictionary for Appointment 31](#_Toc89640438)

[5.3.5 Data Dictionary for Statistics 31](#_Toc89640439)

[5.3.6 Data Dictionary for FAQs 32](#_Toc89640440)

[5.3.7 Data Dictionary for Doctor: 32](#_Toc89640442)

[5.3.8 Data Dictionary for Patient: 33](#_Toc89640444)

[5.3.9 Data Dictionary for Admin: 33](#_Toc89640445)

[5.3.10 Data Dictionary for Payment Plan: 34](#_Toc89640446)

[5.3.11 Data Dictionary for Payments: 34](#_Toc89640447)

[6. Human Interface Design 34](#_Toc89640448)

[6.1 Screen Images 34](#_Toc89640449)

[6.1.1 Admin Dashboard: 35](#_Toc89640450)

[6.1.2 Home 36](#_Toc89640451)

[6.1.3 Neglect Detector Screen: 37](#_Toc89640452)

[6.1.4 Signup 38](#_Toc89640453)

[6.1.5 Rehabilitation Screen: 38](#_Toc89640454)

[6.1.6 Login Page: 39](#_Toc89640455)

[6.1.7 Screen Objects and Actions 40](#_Toc89640456)

[7. Plagiarism Report: 41](#_Toc89640457)

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** |
| **Ms. Isma ul Hassan 's comments** Overall idea of the project is interesting. Good presentation. Patient or his/her guardian should also be able to view the reports. Revise the use case diagram | Guardian actor has been added. Guardian can view patient’s reports. Use case diagrams have also been revised. |
| **Mr. Zaheer ul Hussain Sani 's comments** Suggested to have parents account to see the progress. App should be HIPAA compliant. | Guardian actor has been added and can view patient’s progress. We have made the required changes in the SRS. We are using bcrypt node encryption library for data encryption, after permission from Sir Zaheer. |
| **Mr. Basharat Mehmood 's comments** Overall idea is interesting, and scope is ok. Students were well prepared. However, it is suggested that enrich the patient module by adding features such as registration, view progress reports. Use-case diagram needs minor changes (as suggested during the presentation). Revise the SRS document after adding features. | Guardian actor has been added. We have made the required changes in the SRS. |

Supervised by

Mr. Khurram Iqbal

Signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Introduction

The proposed system shall be web application designed for doctors to assess and rehabilitate patients suffering from Spatial neglect. Neglect detector will detect neglect in patients through examining exercise result and then Rehabilitation plan will be provided that will consist of series of exercises. The doctor will be able to keep a log of his patients and schedule appointments, where in each appointment, the patient will perform the required exercises. The doctor will be able to generate reports and visualize them to understand the severity of the disorder of the patient. The guardian will also be able to track progress of his/her patients.

The doctor and guardian can login and signup. Every time they signup an email is sent to verify the account. They can also login through their google accounts. Each can view their dashboards. They can view, edit their profile. They can also reset their passwords.

The doctor can post a review for the app. The admin can respond to it. Guardian can start the detector and view the whiteboard. Marker color and size can be changed, white board can be cleared, and strokes on the whiteboard can be undone.

Next, we are moving towards other functionality like rehabilitation and reports related stuff.

# Design Methodology and Software Process Model

Neglect app will be developed using incremental process model as it allows us to develop the application in form of chunks and with development of each workable chunk project progress and we can reach our requirements within due time. Incremental models allow us to prioritize our specific requirements and allow break down of tasks.

Design methodology will be Object Oriented Methodology as it allows separation of duties easily and provide us cost effective maintainability, usability and fills the gap between problem and solution. Also, this approach is near to real word so makes us understand the system in easy way.

# System Overview

Neglect App will be a platform for Doctors, Guardians and Patient. Doctors will be able to easily track the performance of patients through generated reports. Future decision making will be easy for them as well. Guardians will be able to add patients and request for Appointments also they can keep track of patient performance through visualizing reports. Patients will be able to perform Neglect detection and rehabilitation exercises and their performance will be measured. it’s all about Neglect community building.

MVC Architectural Design Pattern will be followed for Neglect App.

## Architectural Design

### Box and Line Diagram:

**Diagram

Description automatically generated**

**Fig 1: Box and Line Diagram**

### Architecture Diagram:

Diagram

Description automatically generated

**Fig 2: Architecture Diagram**

### MVC Components:

#### Views:

Views are what is visible to the client. Clients interact with Views. Views are controlled by Controllers.

Views are UI of an application. Through View layer, Data is exposed to Client. In our application Views will be related to Guardian, Doctors, Detector, Rehabilitator, Reviews etc.

#### Controller:

Controllers basically controls View visibility and interacts with the Models to get State and update State. Controllers are extremely important in dealing with User authentication as it authenticates correct users. Controllers that’s help of utilities like middleware for complex works and break task into sub tasks.

Our application will be consisted of Users controllers.

#### Models:

Models hold the state. It basically updates a View. Model also applies set of rules on data to be stored correctly.

Models contain schema to be enforce on Database. Our application will contain models related to Users, Reviews, FAQs, statistics etc.

# Design Models

## Activity Diagrams:

### Activity Diagram for Detector:

**Fig 3: Activity Diagram for Detector**

Diagram

Description automatically generated

### Activity Diagram for Schedule Appointment:

Diagram

Description automatically generated

**Fig 4: Activity Diagram for Appointments**

### Activity Diagram for Review:

Diagram

Description automatically generated

**Fig 4: Activity Diagram for Reviews**

### Activity Diagram for Rehabilitator:



**Fig 5: Activity Diagram for Rehabilitator**

### Activity Diagram for Post request:



**Fig 6: Activity Diagram for Requests**

### Activity Diagram for Respond to Request:



**Fig 7: Activity Diagram for Requests**

### Activity Diagram for Payments:



**Fig 8: Activity Diagram for Payments**

### Activity Diagram for Report Generator:



**Fig 9: Activity Diagram for Report Generator**

## Sequence Diagrams:

### Sequence Diagram for Scheduling Appointments:



**Fig 10: Sequence Diagram for Appointments**

### Sequence Diagram for Detector:



**Fig 11: Sequence Diagram for Detector**

### Sequence Diagram for Rehabilitator:



**Fig 12: Sequence Diagram for Rehabilitator**

### Sequence Diagram for Patient Logging:



**Fig 13: Sequence Diagram for Patient Logging**

### Sequence Diagram for Report Generator:



**Fig 14: Sequence Diagram for Report Generator**

**Class Diagram:**

**Diagram, schematic

Description automatically generated**

Diagram

Description automatically generated

**Fig 18: Class Diagram**

# Data Design

* We are using NoSQL Database called MongoDB.
* So, we have Collections inside our database and each Collection is referred to entity that holds its existence.
* We are following BSON format for storing our data and Schema will be Mongoose Schema which is library for Object Data Modeling for MongoDB to enforce schema.
* MongoDB, Mongoose Query will be used to process the data.
* Data will be organized in BSON documents inside collections and each document is one separate record.

Diagram

Description automatically generated

**Fig 19: Data design**

## Major Collections:

1. Doctor
2. Patient
3. Guardian
4. Review
5. Reply
6. Admin
7. Payment Plan
8. Appointment
9. Payment
10. Statistics
11. FAQs

## Data Structures:

### Data Structure for Guardian:

const mongoose=require('mongoose');

const guardianSchema=new mongoose.Schema(

{

name:{

type:String,

required:[true,"Please Enter Your Name"],

trim:true

},

age:{

type:Number,

required:[true,"Please enter Your Age"]

},

email:{

type:String,

required:[true,"Please Enter Your Email"],

trim:true,

unique:true

}

,

password:{

type:String,

required:[true,"Please Enter Passsword"],

},

gender:{

type:String,

required:[true,"Please Enter Your Gender"]

},phonenumber:{

type:String,

},

address:{

type:String

},country:{

type:String

},

city:{

type:String

}

, profilepic:{

type:String,

default:"https://res.cloudinary.com/drimnkool/image/upload/v1636226843/samples/landscapes/beach-boat.jpg"

}

},

{

timestamps:true

}

)

module.exports=mongoose.model("Guardian",guardianSchema);

### Data Structure for Reviews:

const mongoose=require('mongoose');

const reviewSchema=new mongoose.Schema(

{

userid:{

type:mongoose.Types.ObjectId,

required:true

},

usertype:{

type:String,

required:true

}

,name:{

type:String

},

subject:{

type:String,

required:true

},

comment:{

type:String,

required:true

}

},

{

timestamps:true

}

)

module.exports=mongoose.model("Review",reviewSchema);

### Data Structure for Review Reply:

const mongoose=require('mongoose');

const reviewReplySchema=new mongoose.Schema(

{

reviewid:{

type:mongoose.Types.ObjectId,

ref:'Review'

},

reply:{

type:[{

userid:{type:mongoose.Types.ObjectId},

usertype:{type:String},

comment:{type:String}

}]

}

},

{

timestamps:true

}

)

module.exports=mongoose.model("Reviewreply",reviewReplySchema);

### Data Structure for Appointment:

const mongoose=require('mongoose');

const appointmentSchema=new mongoose.Schema(

{

doctorid:{

type:mongoose.Types.ObjectId,

ref:'Doctor'

},

patientid:{

type:mongoose.Types.ObjectId,

ref:'Patient'

},

date:{

type:Date

},

starttime:{

type:String

},

endtime:{

type:String

}

},

{

timestamps:true

}

)

module.exports=mongoose.model("Appointment",appointmentSchema);

### Data Structure for Statistics:

const mongoose=require('mongoose');

const statSchema=new mongoose.Schema(

{

appointmentid:{

type:mongoose.Types.ObjectId,

ref:'Appointment'

},

patientid:{

type:mongoose.Types.ObjectId,

ref:'Patient'

},

score:{

type:[{

round1:{type:Number},

round2:{type:Number},

round3:{type:Number}

}]

}

},

{

timestamps:true

}

)

module.exports=mongoose.model("Statistic",statSchema);

### Data Structure for FAQ:

const mongoose=require('mongoose');

const faqSchema=new mongoose.Schema(

{

question:{

type:String,

required:[true,"Please Enter Question"]

},

answer:{

type:String,

required:[true,"Please Enter Answer"]

}

},

{

timestamps:true

}

)

module.exports=mongoose.model("Faq",faqSchema);

### Data Structure for Doctor:

var mongoose = require('mongoose');

const doctorSchema=new mongoose.Schema(

{

name:{

type: String,

required: true

},

age:{

type: Number,

default:25

},

email:{

type: String,

unique: true,

required: true

},

password:{

type: String,

required: true

},

gender:{

type: String,

default:"male"

},

phonenumber:{

type: String,

default:"1234567"

},

address:{

type: String,

default:"hello address"

},

qualification:{

type: String,

default:"abc"

},

paymentStatus:{

type: Number,

default:0

},

country:{

type: String,

default:"Pakistan"

},

city:{

type: String,

default:"Rwp"

},

profilepic:{

type: String,

default:"https://res.cloudinary.com/drimnkool/image/upload/v1636226843/samples/landscapes/beach-boat.jpg"

},

planId:{

type: mongoose.Types.ObjectId,

ref: 'PaymentPlan'

},

paymentStatus:{

type: Number

},

role:{

type: Number,

default:2

},

patients: {

type: [{

sid: {

type: mongoose.Types.ObjectId,

ref: 'Patient'

}

}]

}

}

)

module.exports=mongoose.model("Doctor",doctorSchema);

### Data Structure for Patient:

var mongoose = require(mongoose);

const patientSchema=new mongoose.Schema(

{

name:{

type: String,

required: true

},

age:{

type: Number,

required: true

},

neglectType:{

type: String,

required: true

},

gender:{

type: String,

required: true

},

phoneNumber:{

type: String,

required: true

},

address:{

type: String,

required: true

},

country:{

type: String,

required: true

},

city:{

type: String,

required: true

},

lastUpdated: {

type: String,

format: date

},

currentMilestone: {

type: Number

},

milestone: {

type: Number

},

guardianId:{

type: mongoose.Types.ObjectId,

ref: 'Guardian',

required: true

},

}

)

module.exports=mongoose.model("Patient", patientSchema);

### Data Structure for Admin:

var mongoose = require(mongoose);

const adminSchema=new mongoose.Schema(

{

name:{

type: String,

required: true

},

age:{

type: Number,

required: true

},

email:{

type: String,

unique: true,

required: true

},

password:{

type: String,

required: true

},

gender:{

type: String,

required: true

},

phoneNumber:{

type: String,

required: true

},

experience: {

type: [{

company: {

type: String,

required: true

},

years:{

type: Number,

required: true

}

}]

},

profilePic:{

type: String,

required: true

},

role: {

type: String,

default: 3

}

}

)

module.exports=mongoose.model("Admin", adminSchema);

### Data Structure for Payments Plan:

var mongoose = require(mongoose);

const paymentplanSchema=new mongoose.Schema(

{

name:{

type: String,

required: true

},

price:{

type: Number,

required: true

},

description:{

type: String,

required: true

},

duration: {

type: Number,

required: true

}

}

)

module.exports=mongoose.model("PaymentPlan", paymentplanSchema);

### Data Structure for Payments:

var mongoose = require(mongoose);

const paymentSchema=new mongoose.Schema(

{

doctorId:{

type: mongoose.Types.ObjectId,

ref: 'Doctor',

required: true

},

paidAmount: {

type: Number,

required: true

},

paydate:{

type: Date

}

}

)

module.exports=mongoose.model("Payment", paymentSchema);

## Data Dictionary

### Data Dictionary for Guardian Account

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| address | String | Guardian current Address |
| age | Number | Guardian Age |
| city | String | City where Guardian Lives |
| country | String | Country |
| email | String | Guardian Email for Account |
| gender | String | Guardian Gender |
| name | String | Guardian Name |
| password | String | Guardian Account Password |
| phonenumber | String | Guardian Contact Number |
| profilepic | String | Profile picture for guardian account. |

**Table 1: Guardian Account**

### Data Dictionary for Reviews

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| Userid | Schema.Types.objectId | Foreign key references Guardian, Admin, Doctor |
| usertype | String | Type of user posted a review |
| Subject | String | Review Subject as title |
| comment | String | A review description or body |

**Table 2: Reviews**

### Data Dictionary for Reviews reply

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| comment | String | Description or body of reply to review. |
| reply | Array | All replies related to specific review |
| Reply Array is array of BSON Objects and includes | | |
| reviewid | Schema.Types.objectId | Review id against replies are made. It references Review collection. |
| userid | Scheme.Types.objectId | User id of who posted a reply. References to Guardian, Admin, Doctor |
| usertype | String | Type of user posted a review reply |

**Table 3: Reviews Reply**

### Data Dictionary for Appointment

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| date | Date | Schedule Appointment Date |
| Doctored | Schema.Types.objectId | Id of Doctor who is scheduling Appointment and it references Doctor collection. |
| endtime | TimeStamp | Ending time for Appointment |
| patientid | Schema.Types.objectId | Patient id for the patient whose appointment will be scheduled. It references Patient collection and patient is also in patient list for doctor. |
| starttime | TimeStamp | Ending Time for Appointment |

**Table 4: Appointments**

### Data Dictionary for Statistics

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| appointmentid | Schema.Types.objectId | Appointment id references to Appointment Collection. |
| patientid | Schema.Types.objectId | Patient id whose Appointment stats needs to be shown. |
| round1 | Number | Round1 Score |
| round2 | Number | Round2 Score |
| round3 | Number | Round3 Score |
| score | Array | Score array for storing round scores. |
| Scores array is collection of round scores and include round1, round2 and round3 score. | | |

**Table 5: Statistics**

### Data Dictionary for FAQs

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| answer | String | FAQ answer to be posted |
| question | String | FAQ Question to be posted. |

### 

**Table 6: FAQ**

### Data Dictionary for Doctor:

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| name | String | Doctor’s name |
| age | Number | Doctor’s age |
| email | String | Doctor’s email |
| password | String | Doctor’s account password |
| gender | String | Doctor’s gender |
| phoneNumber | String | Doctor’s contact number |
| address | String | Doctor’s current address |
| qualification | String | Doctor’s qualification |
| country | String | Doctor’s residential country |
| city | String | Doctor’s residential city |
| profilePic | String | Doctor’s profile picture |
| paymentStatus | Number | Doctor’s payment status |
| planID | Schema.Types.objectId | Doctor’s payment plan |
| patients | Array | Doctor’s patients |
| role | Number | User role |

### 

**Table 7: Doctor**

### Data Dictionary for Patient:

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| name | String | Patient’s name |
| age | Number | Patient’s age |
| neglectType | String | Patient’s neglect type |
| gender | String | Patient’s Gender |
| phoneNumber | String | Patient’s Contact Number |
| address | String | Patient’s current Address |
| currentMilestone | Number | Patient’s milestone reached |
| Milestone | Number | Patient’s milestone |
| lastUpdated | String | Date of milestone reached |
| country | String | Patient’s residential country |
| city | String | Patient’s residential city |

**Table 8: Patients**

### Data Dictionary for Admin:

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| name | String | Admin’s name |
| age | Number | Admin’s age |
| email | String | Admin’s email |
| password | String | Admin’s password |
| gender | String | Admin’s Gender |
| phoneNumber | String | Admin’s Contact Number |
| Experience | Array | Admin’s work experience |
| profilePic | String | Admin’s profile picture |
| Role | Number | User role |

**Table 9: Admin**

### Data Dictionary for Payment Plan:

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| Name | String | Plan’s name |
| Price | Number | Plan’s price |
| Description | String | Plan’s description |
| Duration | String | Plan’s duration |

**Table 10: Payment Plan**

### Data Dictionary for Payments:

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | Schema.Types.objectId | Primary key |
| doctorID | Schema.Types.objectId | Foreign Key |
| payDate | String | Date doctor paid |
| paidAmount | Number | Amount doctor paid |

**Table 11: Appointments**

# Human Interface Design

## Screen Images

### Admin Dashboard:

Chart, treemap chart

Description automatically generated

**Fig 20: Admin Dashboard**

### Graphical user interface, website Description automatically generatedHome

**Fig 21: Home**

### Neglect Detector Screen:

Graphical user interface, text, application

Description automatically generated

**Fig 22: Detector**

### Signup

Graphical user interface, website

Description automatically generated

**Fig 22: Signup**

### Rehabilitation Screen:

Shape

Description automatically generated

**Fig 23: Rehabilitation**

### Login Page:

Graphical user interface, website

Description automatically generated

**Fig 24: Home**

### Screen Objects and Actions

**Admin dashboard is consisted of options.**

User management where admin will be able to deactivate any User account etc.

Profile Management where admin will be able to manage his profile related work.

FAQ management where admin will be able to manage FAQs appears on website.

Review management where admin will be able to reply to queries posted.

**Doctor Dashboard is consisted of options:**

Profile section where doctor will be able to manage profile.

Patient section where doctor will be able to add or delete patient.

Report section where doctor will be able to keep track of performance of patient.

Appointment section where doctor will be able to keep track of appointments and Patients.

**Guardian Dashboard is consisted of options:**

Appointment section where guardian will be able to request doctor for appointments.

Patient section where guardians can add patients, update etc.

Report section where guardian will be able to keep track of his related patient improvement.

**Detector Screen:**

Detector screens have Image to be copy and whiteboard component where patient will copy an image. Series of drawing related button objects will be able like undo last action, clear board, color picker etc.

**Rehabilitator Screen:**

Rehabilitator screen have image to be scratched along with button objects related to scratching like undo, clear etc.

**Report Screen:**

Report screen have many options through which report can be generated and viewed.

Doctor, Guardian will be able to select specific patient for report also they can generate generic report for patient as compared to other patients. Both statistical, graphical reports can be generated.

**Review Screen:**

Review screen will display all the reviews to User and users can view, post review also they can edit and reply to review.

# Plagiarism Report:

Graphical user interface, text, application

Description automatically generated