

# GoodMead Hospital Management System

From Analysis to Design and Architecture

Sabbir Ahmed, Kristy Cappelli, & Chuck Norris



EN.605.601: Foundations of Software Engineering  
Johns Hopkins University  
November 30, 2020

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
	Figure: Use Case Diagram . . . . .	2
<b>2</b>	<b>Analysis</b>	<b>2</b>
2.1	Activity Diagram . . . . .	2
	Figure: Activity Diagram . . . . .	3
<b>3</b>	<b>Design</b>	<b>4</b>
3.1	Entity Class Diagrams . . . . .	4
	Figure: Entity Class Diagram . . . . .	4
3.1.1	CredentialManager . . . . .	4
3.1.2	DatabaseManager . . . . .	4
3.1.3	DesktopClient . . . . .	5
3.1.4	ClientView . . . . .	5
3.1.5	DoctorView . . . . .	5
	Figure: Database/Record Class Diagram . . . . .	5
3.2	Pseudocode . . . . .	5
<b>4</b>	<b>System Architecture</b>	<b>7</b>
4.1	Component Diagram . . . . .	7
4.1.1	Cloud . . . . .	7
4.1.2	Database . . . . .	7
4.1.3	Authentication Server . . . . .	7
4.1.4	Desktop Client . . . . .	7
4.1.5	Administrative Suite Server . . . . .	7
4.1.6	HMS_webserver . . . . .	7
4.1.7	Mobile Client . . . . .	8
	Figure: Component Diagram . . . . .	8
4.2	Component Entity Mapping . . . . .	9
	Figure: Component-Entity Class Map . . . . .	9
4.3	System Architecture Diagram . . . . .	10
	Figure: System Architecture Diagram . . . . .	10
4.4	UI Flow . . . . .	10
	Figure: Patient UI Frames . . . . .	10
	Figure: Patient UI Flow . . . . .	14
<b>A</b>	<b>Code snippets</b>	<b>15</b>
<b>B</b>	<b>Mobile Application Prototype</b>	<b>20</b>

# 1 Introduction

The purpose of this deliverable is to document the analysis, design and system architecture of a use case of **GoodMead: A Hospital Management System (HMS)**. The use case chosen was "Doctor Updates Patient Records" replicated below.

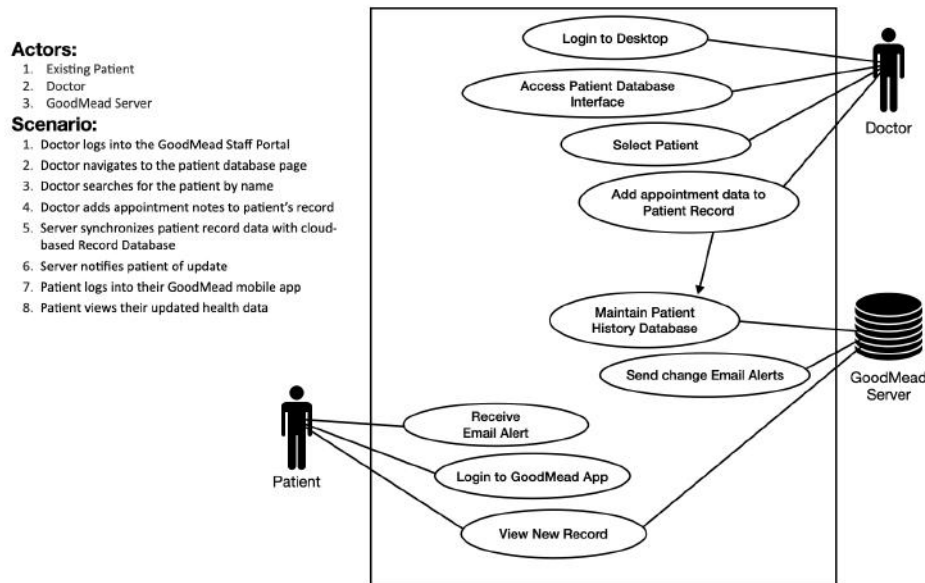


Figure 1: Use Case Diagram

The analysis portion of the deliverable consists of activity diagrams as well as identifies the entity classes of the specified use case. The design portion expands on the entity classes through component diagrams and relevant pseudocode. Lastly, the system architecture portion consists of several system architecture diagrams on various layers of details as well as UI-Flow diagrams corresponding to an Actor using the interfaces identified in the use case.

## 2 Analysis

The use case we chose to revisit was our second: A doctor updating a patient's record. This use case demonstrates the interface between doctors and patients, drawing information and services from multiple backend servers and databases. It addresses two major categories of stakeholders for this software - the patient as a hospital client and the doctor as a staff member who relies on this software daily.

This is a process that will occur after each patient interaction by a doctor, surgeon, or other hospital technician. Data integrity and reliability are needed to comply with laws and regulations about hospital liability and patient medical data. The data update process cannot lose or conflate data and maintains information necessary for excellent treatment, billing, and liability. The database hosting infrastructure must have satisfactory uptimes, periodic backups, and meet laws surrounding the security and privacy of health information. As such, this is an exemplary use case to provide coverage of entity classes, each system component's interactions, and further design sketches.

### 2.1 Activity Diagram

Following is an Activity Diagram extending this Use Case which was generated using the tool draw.io.

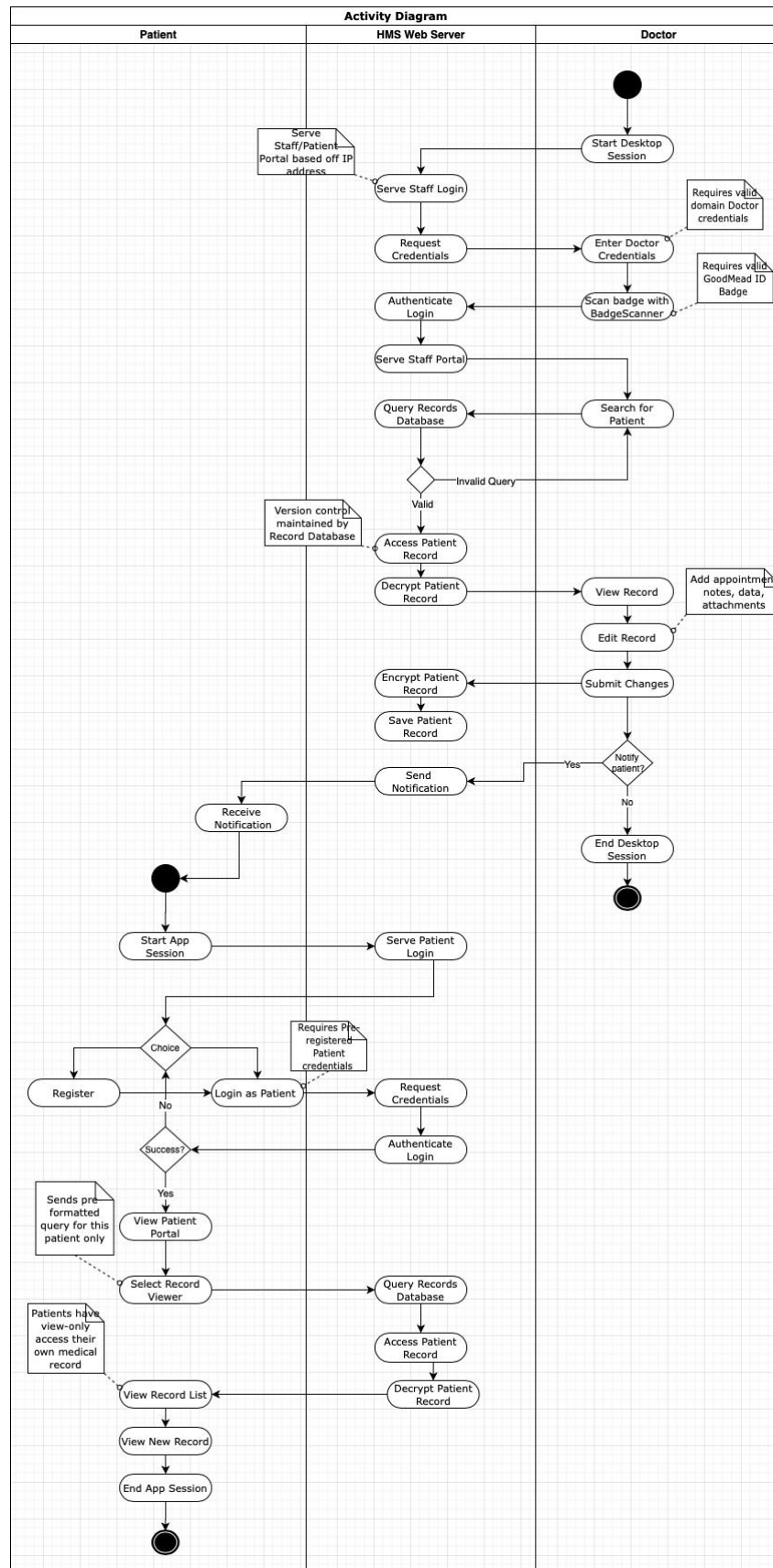


Figure 2: Activity Diagram

**Actors:** Patient, HMS Web Server, Doctor

Both the Doctor and Patient interact through a portal served by the HMS Web Server. This server interacts with other backend servers and databases through their exposed APIs detailed in the Design and Architecture sections. The Doctor Actor includes using a badge scanner device for 2-Factor Authentication.

### 3 Design

The design portion of the deliverable expands on the entity classes detailed in the analysis of the HMS.

#### 3.1 Entity Class Diagrams

The classes include the database classes and design. The design of the system follows a facade pattern by concealing the complexities of the interconnected classes through high-level application programming interfaces (API).

Figure 3 extends and expands upon the original classes.



Figure 3: Entity Class Diagram

##### 3.1.1 CredentialManager

The **CredentialManager** class manages the user authentication and sessions. The class provides high-level access to authenticate user login information as well as update users' credentials. Its attributes **username** and **session\_id** are protected because they are shared amongst the classes that instantiate it. All the other attributes are private.

##### 3.1.2 DatabaseManager

The **DatabaseManager** class manages the record classes defined in Figure 4. The class provides high-level access to create, read, update and read from all of the records in the system.

Additional operations such as `commit()` and `set_permission(permission: Integer)` are provided for additional security for the database.

### 3.1.3 DesktopClient

The `DesktopClient` class provides high-level access to interact with the `CredentialManager` and `DatabaseManager` classes. The class can be inherited by a user interface engine such as a graphical user interface or a web application. The class is intended to be abstract and only exists to be derived for implementation.

### 3.1.4 ClientView

The `ClientView` class inherits from the `DesktopClient` base class. The class implements additional operations to provide the clients (or patients) access to features like updating their contact information or looking up their doctors.

### 3.1.5 DoctorView

The `DoctorView` class inherits from the `DesktopClient` base class. The class implements additional operations to provide the doctors access to features like creating and updating their patients' visit records.

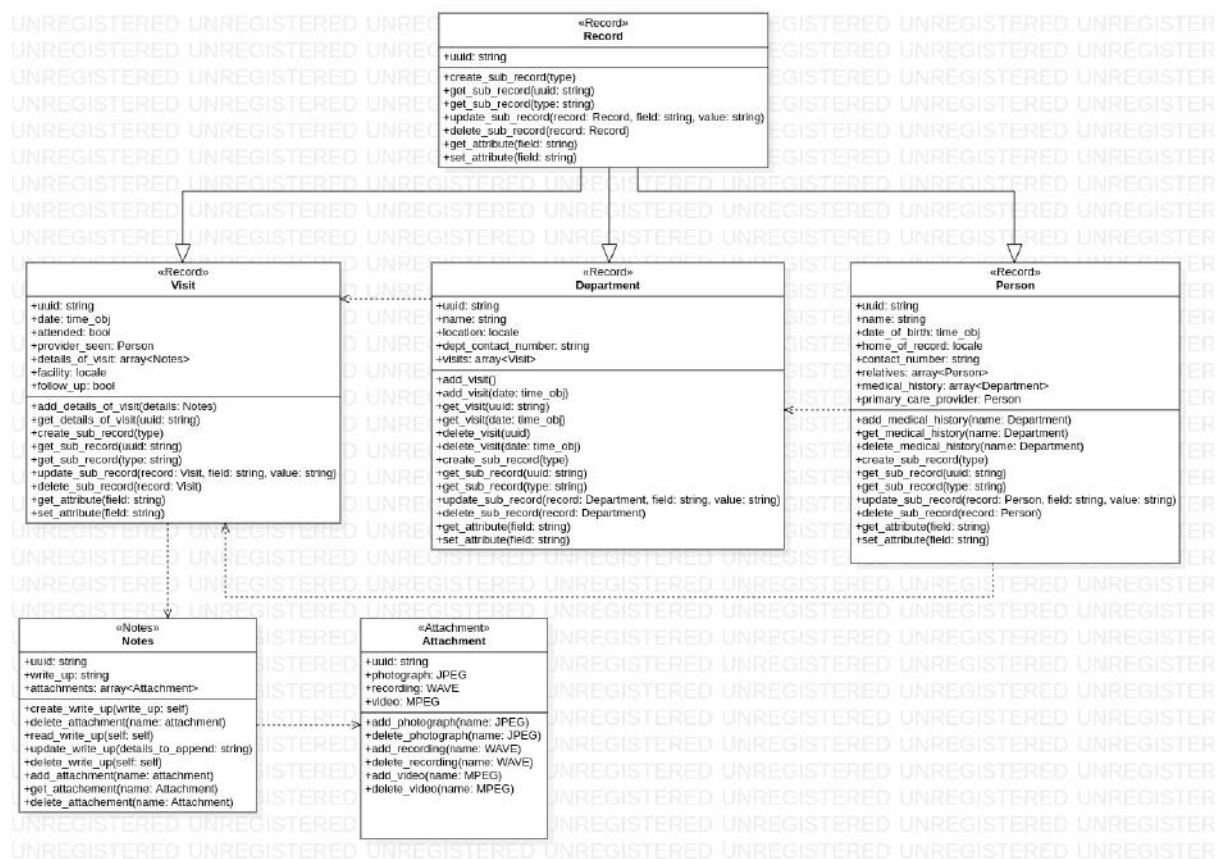


Figure 4: Database/Record Class Diagram

## 3.2 Pseudocode

Some pseudocode of the aforementioned major entity classes has been generated to demonstrate the design choices. The pseudocode snippets can be found in Appendix A. The snippets

do not follow the paradigms or syntax of any existing programming languages.

Some operations used in the snippets are assumed to be builtin functions and are therefore not defined. These operations include:

- `encrypt`
- `decrypt`
- `reference_to`
- `dereference`
- `parse`
- `convert`

Some datatypes used in the snippets are also included without implementation. These datatypes include:

- `PatientRecord`
- `VisitRecord`
- `QueryMap`
- `AdministrativeSuite`

## 4 System Architecture

### 4.1 Component Diagram

The components of the system are connected (networked, interface, etc) to create the GoodMead HMS. These connections are shown in Figure 5. Each component is connected to other components through a port. This port is connected to some interface which allows it to communicate with other connected components. The interface represents interactions with the APIs described in the Section 3.1. The major components of the system are the **Cloud**, **HMS\_webserver**, **Authentication Server**, **Administration Suite**, **Desktop Client**, **Badge Reader**, and **Mobile Client**.

#### 4.1.1 Cloud

The **Cloud** component hosts the **Database** which is where hospital records will be stored. The cloud provider is not a part of the design. Considerations for the cloud hosting service would be discussed with stakeholders.

#### 4.1.2 Database

The **Database** provides access to patient records to authorized users. An encryption component is provided by the database as well to ensure that patient records are not view-able at rest.

#### 4.1.3 Authentication Server

**Authentication Server** which provides session tokens and verification that login sessions are authorized to have access to portions of the HMS system such as the records database. This component connects, via the **auth\_interface**, to the **Cloud→Database**, **Desktop Client**, and **HMS\_webserver** components to provide the GoodMead HMS with role base access controls.

#### 4.1.4 Desktop Client

The **Desktop Client** is where administrative staff can use software to manage patient records, schedule appointments, and chat with patients and staff. This component is connected to the **HMS\_webserver** to provide user portal as discussed in the section **HMS\_webserver**. As an additional layer of on site security we have a **Badge Reader** component to further verify staff personnel access.

#### 4.1.5 Administrative Suite Server

Any staff member that uses the **Desktop Client** will have access to a suite of administrative tools via the **Administrative Suite Server** component which will provide access to Email and Calendar (shown on the diagram) as well as other. The Email component on the Administrative Suite Server also ties into the **Desktop Client** to send out email notifications to personnel when they are scheduled for appointments or their records are updated.

#### 4.1.6 HMS\_webserver

The **HMS\_webserver** hosts a website and and user portal which will provide different features and capabilities based on the user and role logging in. Two shown views are the staff and patient view. With component is also connected to the internet as it provides an external user portal and experience for third party devices such as a patients mobile device



#### 4.1.7 Mobile Client

Patients can view portions of their records, contact info, appointments, and more via a mobile application (see: Appendix B) installed in a **Mobile Client**. The application uses a webview pattern which renders locally remote content. Connecting to the **HMS\_webserver** through the **Mobile Client** is not substantively different from a desktop client accessing the website through the internet. This facilitates remote access to a subset of the features of the HMS to authorized personnel.

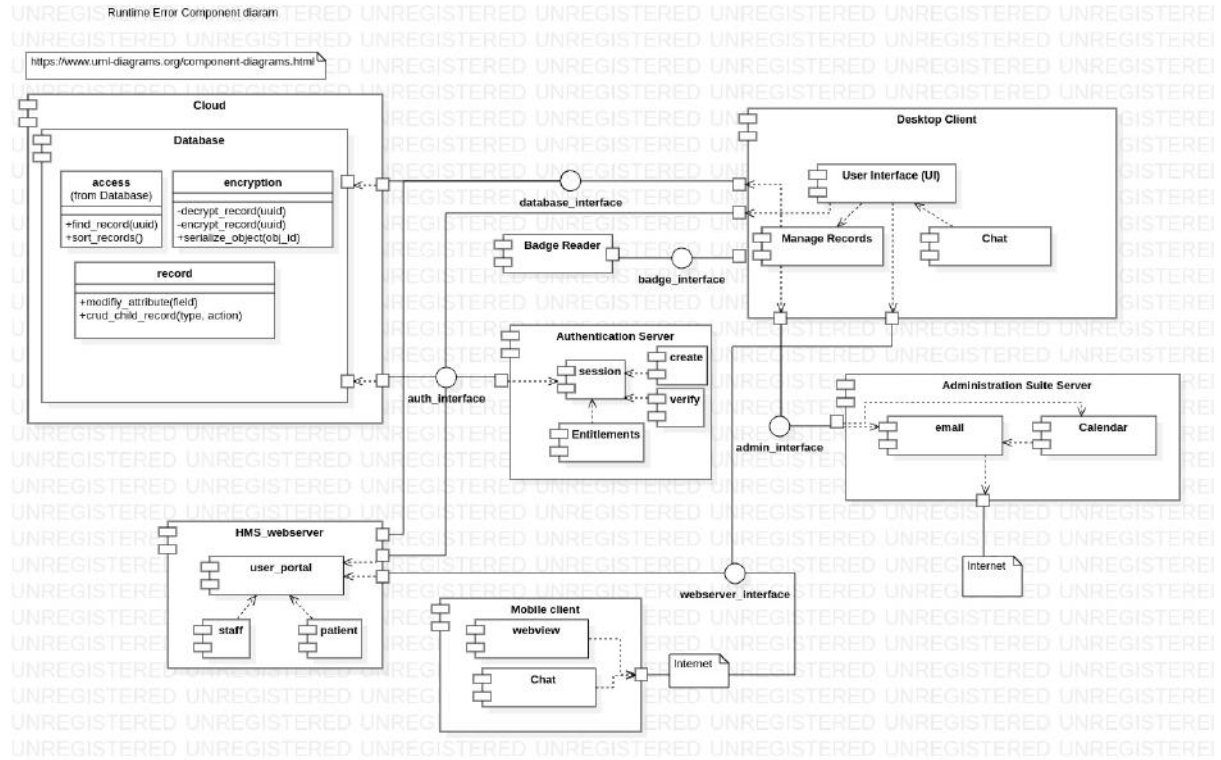


Figure 5: Component Diagram

## 4.2 Component Entity Mapping

The component to entity class mapping allows us, at a glance to verify which components are linked and directly relevant to which entity classes (see: Figures 3 and 4). Most of the components have at least one entity class that it maps to that describes its behaviour at a high level (attributes and functions). There are two components that do not have entity classes that map to them: **Mobile Client** and **Badge Reader**. For the **Badge Reader** we envision a third-party source providing this solution. The **Mobile Client** component utilizes the **Client View** entity class which is served by the HMS WebServer. The component with the most entity classes is the **Cloud** portion of the system. This is because we utilize a database class on the cloud which manages interactions with the multiple record classes which demonstrates the complexity that would be required in a hospital records system.

Component	Entity Classes
Authentication Server	Credential Manager
Desktop Client	DesktopClient
Administration Suite Server	Administration Suite
HMS_webserver -> user_portal	DoctorView, ClientView
Cloud	DatabaseManager, Record, PatientRecord, Department, Visit

Table 1: Component-Entity Classes

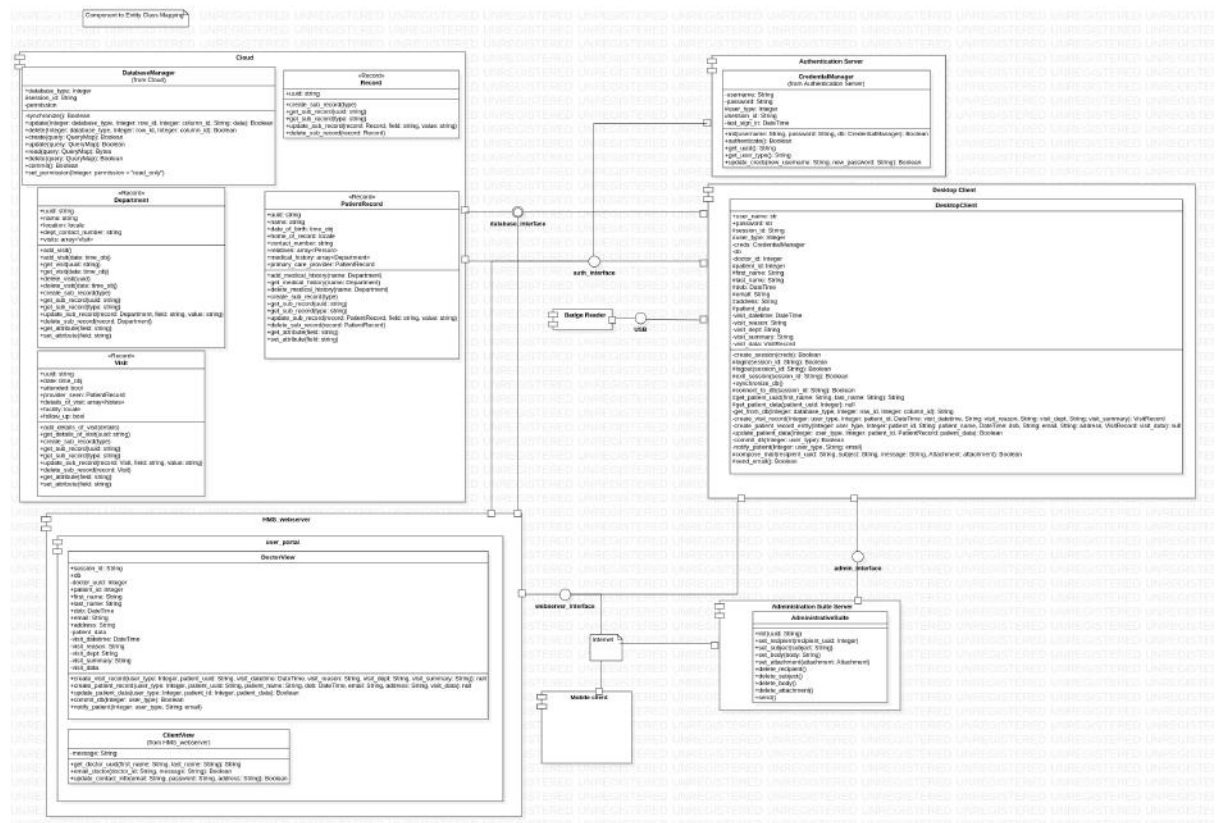


Figure 6: Component-Entity Class Map

### 4.3 System Architecture Diagram

The system architecture diagram of the deliverables draws everything together to show how different portions of the project work relate both in function and in scope. The system architecture diagram shows our different functions or processes on a horizontal axis and scope on the vertical axis, broad to narrow, with the roots being different actions the system might take—considering our use case. Dependencies are shown with dotted arrows. Looking at the system architecture diagram in Figure 7 we can see that the system’s function of updating a patients record depends on four components of the system: **Database**, **Desktop Client**, **AdministrationSuite**, and **AuthenticationServer**. Notice that this diagram also includes the interfaces for some of the components. This demonstrates that in order to access more narrow in scope functions or capabilities one must access through that component’s interface. Interfaces and their connections are also shown in Figure 5.

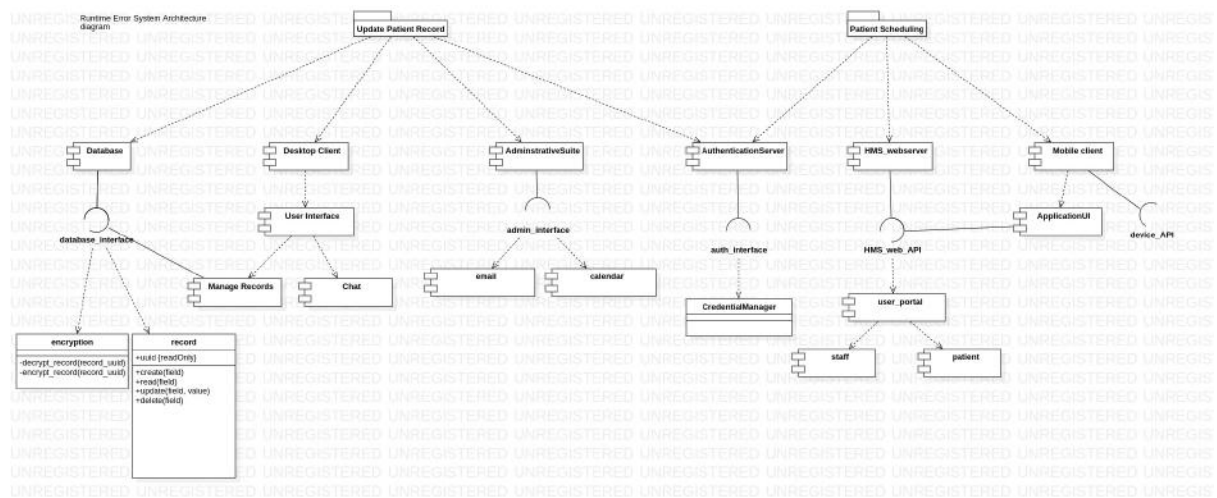


Figure 7: System Architecture Diagram with use-cases as roots

### 4.4 UI Flow

This UI Flow corresponds to the Activity Diagram from the perspective of the Patient Actor using a mobile device with the GoodMead application installed. The patient is an existing patient with a previously registered account with GoodMead Hospital.

1. Patient opens the GoodMead app
2. Patient logs in using credentials they established when they first became a GoodMead patient
3. Patient views their Profile page and checks their listed information
4. Patient taps on the Records link from Dashboard
5. Patient views a list of their Visit Records with GoodMead Hospital and taps on the latest Record, which has a Notification icon marking it as recently edited
6. Patient views the Record from their recent Optometry appointment. The doctor in the Activity Diagram had added Patient Visit Notes to the Record, which the patient can see marked as new with a Notification Icon

We used the prototyping tool Figma to generate the following UI flow images as well as the User App Prototype in Appendix B.

9:27



# GoodMead

LOG IN

REGISTER

9:27



## Log in

jane@example.com

.....

LOG IN

q w e r t y u i o p

a s d f g h j k l



z

x

c

v

b

n

m



123

space

return



9:27



# Dashboard

## QUICK LINKS



Urgent Care



Appointments



Insurance



Records



Prescriptions

PROFILE



9:27



# GoodMead

Welcome, Jane



Jane Smith

MEMBER SINCE 2020

EDIT PROFILE

PATIENT HISTORY

INSURANCE CARD



9:27



## Records

### VISIT HISTORY

#### OPTOMETRY

ID: 08689928

DR. WILL KENNY  
09/15/2020



#### WELLNESS VISIT

ID: 08688201

DR. JAMES SLOANE  
07/21/2020

#### WELLNESS VISIT

ID: 08623682

DR. JAMES SLOANE  
01/06/2020



9:27



## Record

### Optometry



You visited Dr. Will Kenny on  
09/15/2020.

Let us know [how your visit went](#)

Visit ID: 08689928  
Type: Eye Exam  
Date: 09/15/2020  
Attended: Yes

### VISIT INFORMATION

#### VISIT & FOLLOW-UP NOTES

ADDED TODAY

### YOUR FORMS

FORM 109A: PATIENT MEDICAL  
HISTORY  
ADDED 09/15/2020

FORM 774: LABORATORY TEST  
RESULTS  
ADDED 09/16/2020



Figma's prototyping functionality becomes more apparent with the following diagram.

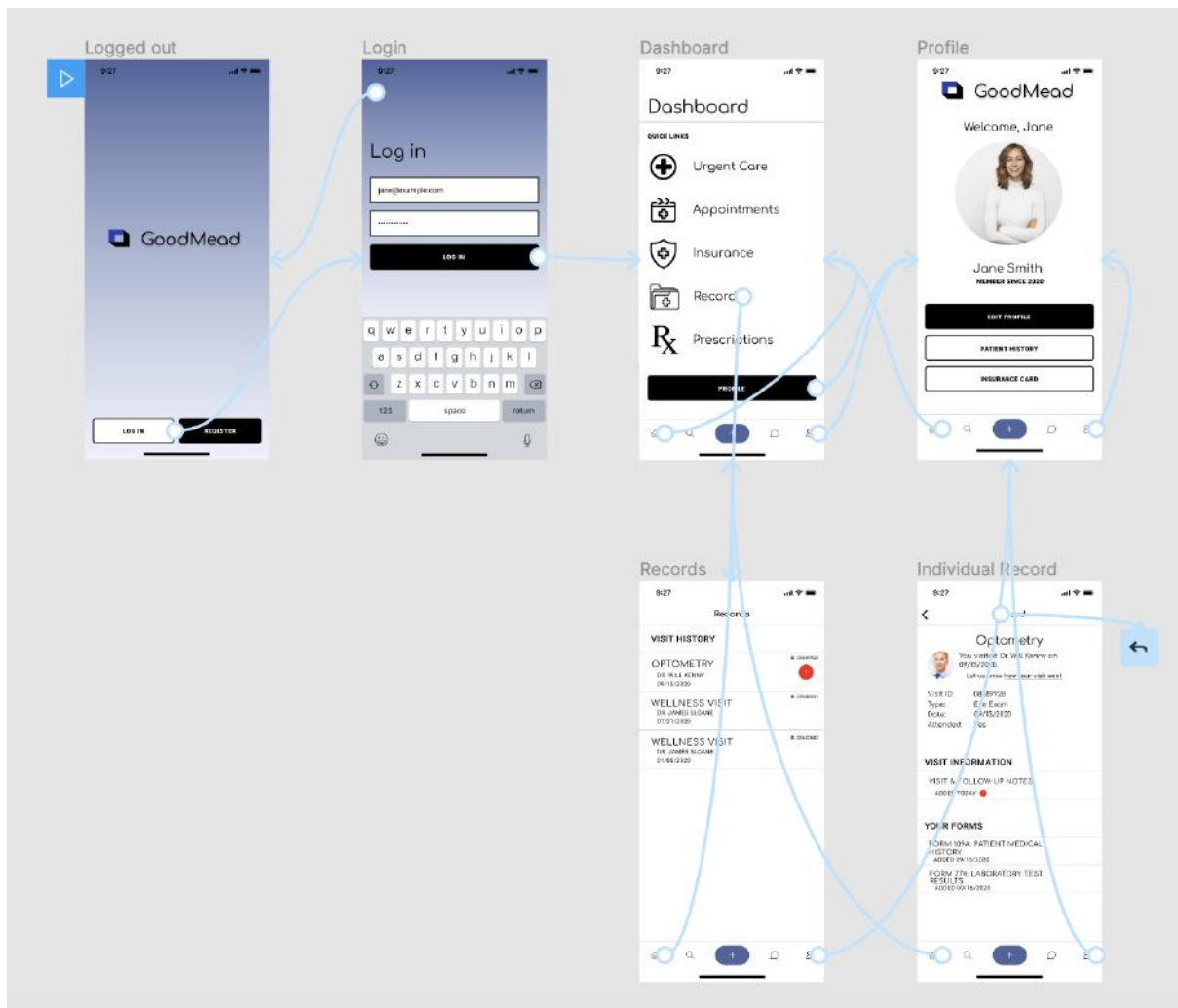


Figure 8: Figure: Patient UI Flow

## Appendix A Code snippets

Pseudocode A.1: CredentialManager Entity Class

---

```
1 class CredentialManager {
2
3   operation init(username: String, password: String, db:
4     CredentialManager) -> Boolean {
5     assign attribute username = encrypt(username)
6     assign attribute password = encrypt(password)
7     assign attribute db = dereference(db)
8     call db.synchronize()
9     if no errors raised {
10       return true
11     } else {
12       return false
13     }
14   }
15
16   operation authenticate() -> Boolean {
17     call db.read(Integer: database_type = PatientRecord, username:
18       Bytes = username, password: Bytes = password)
19     if return value is true {
20       assign attribute uuid = return value uuid
21       assign attribute user_type = return value user_type
22       return true
23     } else {
24       return false
25     }
26   }
27
28   operation get_uuid() -> String {
29     return attribute uuid
30   }
31
32   operation get_user_type() -> String {
33     return attribute user_type
34   }
35
36   operation update_creds(new_username: String, new_password: String)
37     -> Boolean {
38     if user_type is valid {
39       call db.set_permission(Integer: mode = "read_write")
40       call db.update(Integer: database_type = PatientRecord, username
41         : Bytes = new_username, password: Bytes = new_password)
42       call db.set_permission(Integer: mode = "read_only")
43       return true
44     } else {
45       return false
46     }
47   }
48 }
```

---

Pseudocode A.2: CredentialManager Entity Class

---

```
1 class DatabaseManager {
2
```



```

3  operation synchronize() -> Boolean {
4      open database and assign to attribute db
5      synchronize db
6      assign attribute permission to "read_only"
7      if no errors raised {
8          return true
9      } else {
10         return false
11     }
12 }
13
14 operation create(query: QueryMap) -> Boolean {
15     check if attribute permission is valid {
16         parse and convert query into valid executable database query
17         call db.create(query)
18         if no errors raised {
19             return true
20         } else {
21             return false
22         }
23     }
24 }
25
26 operation read(query: QueryMap) -> String {
27     parse and convert query into valid executable database query
28     call db.read(query)
29     if no errors raised {
30         convert return value to String
31     } else {
32         return empty string
33     }
34 }
35
36 operation update(query: QueryMap) -> Boolean {
37     check if attribute permission is valid {
38         parse and convert query into valid executable database query
39         call db.update(query)
40         if no errors raised {
41             return true
42         } else {
43             return false
44         }
45     }
46 }
47
48 operation delete(query: QueryMap) -> Boolean {
49     check if attribute permission is valid {
50         parse and convert query into valid executable database query
51         call db.delete(query)
52         if no errors raised {
53             return true
54         } else {
55             return false
56         }
57     }
58 }
59
60 operation commit() -> Boolean {

```

```

61     call db.commit()
62     if no errors raised {
63         return true
64     } else {
65         return false
66     }
67 }
68
69 operation set_permission(Integer: permission = "read_only"){
70     assign attribute permission = permission
71 }
72
73 }

```

---

### Pseudocode A.3: DesktopClient Entity Class

---

```

1  class DesktopClient {
2
3      operation create_session(creds: CredentialManager) -> Boolean {
4          instantiate CredentialManager object: creds
5          call creds.init(username: String, password: String, reference_to(
6              db): CredentialManager)
7          call creds.authenticate()
8          while return value is false {
9              call creds.init(username: String, password: String,
10                 reference_to(db): CredentialManager)
11              if return value is true {
12                  assign session_id = creds.get_session_id()
13                  assign user_type = creds.get_user_type()
14                  return true
15              }
16          }
17
18      operation connect_to_db(session_id: String) -> Boolean {
19          instantiate DatabaseManager object: db
20          call db.synchronize()
21          call db.create(database_type: Integer = logs, session_id: String,
22             last_login: DateTime = time.now())
23      }
24
25      operation get_patient_uuid(first_name: String, last_name: String) -
26          > String {
27          call db.read(database_type: Integer = PatientRecord, first_name:
28             String = first_name, last_name: String = last_name)
29          return the return value
30      }
31
32      operation get_patient_data(patient_uuid: Integer) -> PatientRecord
33          {
34          call db.read(database_type: Integer = PatientRecord, patient_uuid
35             : String = patient_uuid)
36          return the return value
37      }
38
39      operation compose_mail(recipient_uuid: String, subject: String,
40         message: String, Attachment: attachment) {
41          instantiate AdministrativeSuite object: admin

```

```

35     call admin.init(uuid: String)
36     call admin.set_recipient(recipient_uuid: String)
37     call admin.set_subject(subject: String)
38     call admin.set_body(body: String)
39     call admin.set_attachment(attachment: Attachment)
40 }
41
42 operation send_email() -> Boolean {
43     call admin.send()
44     return the return value
45 }
46 }

```

---

#### Pseudocode A.4: DoctorView Derived Entity Class

---

```

1 class DoctorView inherits from DesktopClient {
2
3     operation create_visit_record(user_type: Integer, patient_uuid:
4         Integer, visit_datetime: DateTime,
5         visit_reason: String, visit_dept: String, visit_summary: String)
6         -> VisitRecord {
7         convert parameters to VisitRecord object
8     }
9
10    operation create_patient_record(user_type: Integer, patient_uuid:
11        String, patient_name: String,
12        dob: DateTime, email: String, address: String, visit_data:
13        VisitRecord) -> PatientRecord {
14        convert parameters to PatientRecord object
15    }
16
17    operation update_patient_data(user_type: Integer, patient_id:
18        String, patient_data: PatientRecord) -> Boolean {
19        call db.update(database_type: Integer = PatientRecords, user_type
20            : Integer = user_type,
21            patient_id: Integer = patient_id, patient_data: PatientRecord =
22            patient_data)
23        call db.commit()
24        call compose_mail(recipient_uuid: String = patient_uuid, subject:
25            String = "Record updated",
26            message: String = "Dear patient, your records have been updated
27                \n")
28        call send_mail()
29        return the return value
30    }
31 }

```

---

#### Pseudocode A.5: ClientView Derived Entity Class

---

```

1 class ClientView inherits from DesktopClient {
2
3     operation get_doctor_uuid(first_name: String, last_name: String) ->
4         String {
5         call db.read(database_type: Integer = DoctorRecords, first_name:
6             String = first_name, last_name: String = last_name)
7         return the return value
8     }
9 }

```

```
7
8  operation update_contact_info(email: String, password: String,
9    address: String) -> Boolean {
10    call create_patient_record(user_type: Integer, patient_uuid:
11      Integer, email: String, address: String)
12    save reference to return object
13    call update_patient_data(user_type: Integer, patient_id: Integer,
14      patient_data: PatientRecord)
15  }
```

---

## **Appendix B    Mobile Application Prototype**

The following pages contain images from the prototype mobile application we generated for this project. The tool Figma allows us to create a UI frames then link them together through defined interactions. Buttons and objects can be configured to do an action such as change to a different linked frame upon interaction. The Figma file format hosted on our shared github generates a interactive phone template in your browser that enables the tappable/clickable interactions we described in the Patient Actor role of our Activity Diagram.

9:27



# GoodMead

LOG IN

REGISTER

9:27



## Log in

jane@example.com

.....

LOG IN

q w e r t y u i o p

a s d f g h j k l



z

x

c

v

b

n

m



123

space

return



9:27



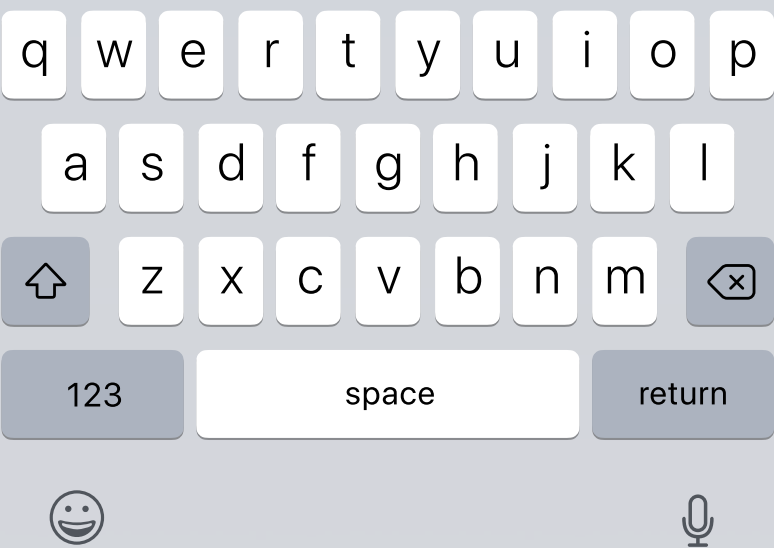
# Register

To register your account, enter the patient account number and password provided with your GoodMead new patient paperwork.

JaneSmith-085383002711

Gzhd7o082hmwp-222jsSD

NEXT



9:27



# Register

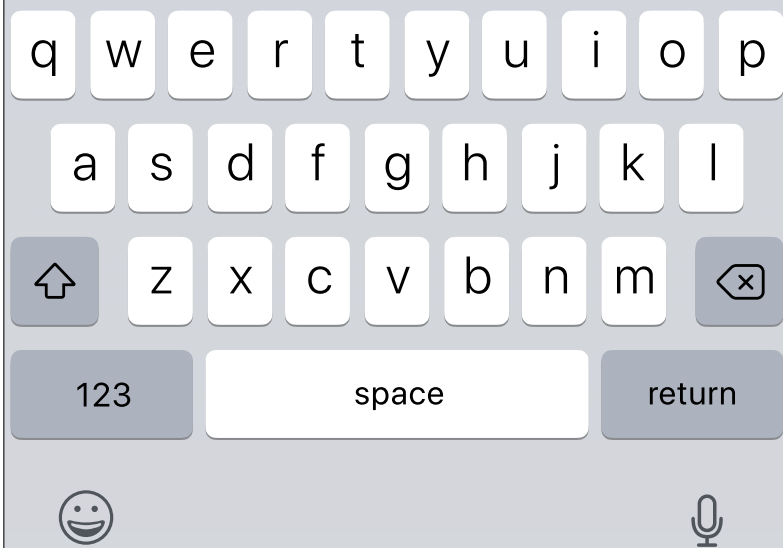
Jane Smith

jane@example.com

.....

SIGN UP

By signing up, you agree to GoodMead's [Terms of Service](#) and [Privacy Policy](#).



9:27



# Dashboard

## QUICK LINKS



Urgent Care



Appointments



Insurance



Records



Prescriptions

PROFILE



9:27



# GoodMead

Welcome, Jane



Jane Smith

MEMBER SINCE 2020

EDIT PROFILE

PATIENT HISTORY

INSURANCE CARD





9:27



Insurance

Insurance Card



EDIT INSURANCE

INSURANCE BENEFITS

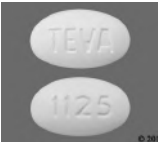


9:27



Prescriptions

JANE SMITH'S MEDICATIONS (1)



SAMPLE DRUG (GENERIC)  
800 MG  
STATUS: PICKED UP

PHARMACY HOME

FIND & PRICE A MEDICATION

LOOK UP MEDICATION INFORMATION,  
COVERAGE, AND PRICING.

PHARMACY LOCATOR

FIND A PHARMACY

MANAGE PRESCRIPTIONS

REFILL HOME DELIVERY



9:27



## Appointments

### Upcoming Appointments

12

APPOINTMENT WITH DR. BETH WILLIAMS  
PODIATRY DEPARTMENT, 11:30 AM

#### November 2020

SUN	MON	TUE	WED	THU	FRI	SAT
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

**SCHEDULE NEW APPOINTMENT**

**MANAGE APPOINTMENTS**



9:27



## Search

Search all pages




9:27

📶

📶


🔋

Chats




Dr. James Sloane

Thank you





Dr. Will Kenny

Hi Jane, please call the office for your test results.





Dr. Beth Williams

See you next week!



+



9:27


📶

📶

🔋


<


Dr. James Sloane





Your prescription should be ready to pick up this afternoon. Let me know if you're not feeling better by Wednesday.

Ok, have a good weekend!







Thank you



+



9:27



## Records

### VISIT HISTORY

#### OPTOMETRY

ID: 08689928

DR. WILL KENNY  
09/15/2020



#### WELLNESS VISIT

ID: 08688201

DR. JAMES SLOANE  
07/21/2020

#### WELLNESS VISIT

ID: 08623682

DR. JAMES SLOANE  
01/06/2020



9:27



## Record

### Optometry



You visited Dr. Will Kenny on  
09/15/2020.

Let us know [how your visit went](#)

Visit ID: 08689928  
Type: Eye Exam  
Date: 09/15/2020  
Attended: Yes

### VISIT INFORMATION

#### VISIT & FOLLOW-UP NOTES

ADDED TODAY

### YOUR FORMS

FORM 109A: PATIENT MEDICAL  
HISTORY  
ADDED 09/15/2020

FORM 774: LABORATORY TEST  
RESULTS  
ADDED 09/16/2020

