What Is a Use Case?

A **use case** is a written or visual description of how a system behaves when used by someone (called an *actor*).

It focuses on **functional requirements**—what the system should do—not how it does it.

 Use cases are often represented in **UML (Unified Modeling Language)** diagrams to show interactions clearly

### Actors

* **Actors** are entities that interact with the system. They can be:
  + **Primary actors**: Initiate the use case (e.g., a customer placing an order).
  + **Secondary actors**: Support the use case (e.g., a payment gateway).
* Actors can be **humans**, **external systems**, or **hardware devices**.

### Use Case Goals

Each use case answers:

* Who is using the system?
* What are they trying to achieve?
* How does the system respond?

Example: In an online shopping system:

* Actor: Customer
* Goal: Place an order
* System: Validates cart, processes payment, confirms ord

### Components of a Use Case

* **Use Case Name**: Clear and goal-oriented (e.g., “Place Order”)
* **Actors**: Who’s involved
* **Preconditions**: What must be true before it starts
* **Main Flow**: Step-by-step interaction
* **Alternate Flows**: Variations or exceptions
* **Post conditions**: What’s true after it ends

### Use Case Diagrams (UML)

* A **use case diagram** visually maps actors to their use cases.
* Key elements:
  + **Ovals**: Use cases
  + **Stick figures**: Actors
  + **Lines**: Relationships between actors and use cases
* Helps stakeholders quickly understand system scope

### Relationships in Use Case Diagrams

* **Association**: Actor ↔ Use case
* **Include**: One use case always triggers another (e.g., “Place Order” includes “Validate Payment”)
* **Extend**: Optional behavior added under certain conditions (e.g., “Track Order” extends “Place Order”)

### Why Use Cases Matter

* Clarify system behavior before coding
* Align developers, designers, and stakeholders
* Help identify edge cases and exceptions
* Form the basis for testing and documentation

**This is the idea that our group extracted from different books to do our project’s use case**

**Core Principles for Our Use Case Design:**

**Zero investment**: Only free, open-source, or free tools with no required payment

**Web-based**: Everything runs in the browser — no native apps or paid APIs

**AI and smart features**: Lightweight, rule-based logic or free AI APIs

**Video conferencing**: Included via free embeddable platforms — no coding required (using iframe

**1. Patient Registration**

**Goals**:

* Create secure patient profiles
* Store demographic and medical history
* Enable role-based access

**Workflow**:

* Patient or receptionist fills out registration form
* System validates and stores data in Firebase
* Firebase Auth ensures secure login and access control

**Success Scenario**:

* Patient profile is created and accessible to authorized users(e.g doctors,nurses)

**Extension Scenario**:

* Duplicate record detected → system prompts merge or alerts admin

**Tech Used:**

* Firebase Firestore (database)
* Firebase Auth (authentication)

### 2. ****Medical Record Update****

**Goals**:

* Let physicians and nurses update patient data
* Maintain accurate, up-to-date health records

**Workflow**:

* Authorized user(physician/nurses) searches for patient
* Edits record and saves changes

**Success Scenario**:

* Record is updated and timestamped

**Extension Scenario**:

* Unauthorized access → system denies update and logs attempt

**Tech Used:**

* Firebase Firestore

### 3. ****Access History via Telemedicine****

**Goals**:

* Enable virtual consultations
* Share medical history during video calls

**Workflow**:

* Patient joins Jitsi Meet session via embedded link
* Physician accesses patient records during call

**Success Scenario**:

* Consultation completed with shared record access

**Extension Scenario**:

* Connection fails → system offers reschedule or offline access

**Tech Used:**

* Jitsi Meet (embedded iframe)

### 4. ****QR Code-Based Record Sharing****

**Goals**:

* Allow patients to share records instantly
* Simplify access for external providers

**Workflow**:

* Patient generates QR code from dashboard
* Provider scans code to view records

**Success Scenario**:

* Record accessed securely via QR

**Extension Scenario**:

* QR expired or invalid → system prompts regeneration

**Tech Used:**

* QRious.js or goqr.me API

### 5. ****Cross-Organization Record Sharing****

**Goals**:

* Share records with clinics, hospitals, labs
* Maintain patient control over shared data

**Workflow**:

* Patient selects records and destination
* System generates secure link and sends it

**Success Scenario**:

Receiving organization accesses records successfully

**Extension Scenario**:

* Link broken or unauthorized → system alerts patient

Tech **Used:**

* Google Drive (link sharing)
* Firebase Hosting

6. **AI-Powered Emergency Alerts**

**Solution**: JavaScript rule-based logic + EmailJS

**Goals**:

* Detect critical health issues
* Notify patients and optionally providers

**Workflow**:

* System scans records for emergency flags
* Sends alert via email or dashboard

**Success Scenario**:

* Patient receives timely alert and takes action

**Extension Scenario**:

* False alert → system logs and allows dismissal

**Tech Used:**

* JavaScript rule-based logic
* EmailJS

### 7. ****Smart Reminders****

**Goals**:

* Remind patients of appointments, medication, checkups
* Support preventive care

**Workflow**:

* Patient sets reminders via form
* System sends email or browser notifications

**Success Scenario**:

* Patient receives reminder and completes task

**Extension Scenario**:

* Reminder missed → system resends or escalates

**Tech Used:**

* EmailJS
* Browser notifications
* Firebase Firestore

### 8. ****Consent-Driven Data Sharing Dashboard****

**Goals**:

* Let patients manage who can access their data
* Provide transparency and control

**Workflow**:

* Patient views sharing permissions
* Toggles access for organizations

**Success Scenario**:

* Permissions updated and logged

**Extension Scenario**:

Unauthorized toggle attempt → system denies and logs

**Tech Used:**

* Firebase Firestore
* HTML dashboard

### 9. ****Manual Vitals Entry / Wearable Sync****

**Goals**:

* Allow patients to input health metrics
* Optionally sync with wearables

**Workflow**:

* Patient enters vitals manually or syncs device
* System stores and analyzes data

**Success Scenario**:

* Metrics saved and used for alerts or trends

**Extension Scenario**:

* Invalid data → system prompts correction

**Tech Used:**

* HTML form
* Optional: Google Fit API

**Tech Integration Summary**

|  |  |
| --- | --- |
| **Feature** | **Free Tool** |
| Database & Auth | Firebase (free tier) |
| Hosting | Firebase Hosting / GitHub Pages |
| Video Conferencing | Jitsi Meet (iframe embed) |
| QR Code | QRious.js or goqr.me API |
| Email Alerts | EmailJS |
| Reminders | Browser notifications |
| Record Sharing | Google Drive / Firebase links |
| AI Logic | JavaScript rules or free OpenAI tier |

## Use Case Diagram

This visual diagram shows the actors and their interactions with the system using standard UML notation.

## Actors in the System

These are the entities (people or systems) that interact with your health platform:

* **Patient**: The primary user of the system
* **Physician**: Medical professional accessing and updating records
* **Receptionist**: Assists with registration and scheduling
* **External Organization**: Clinics, labs, or hospitals receiving shared records
* **System (AI Logic)**: Internal logic that automates alerts and reminders

## Use Cases

These are the system functions that actors interact with:

1. **Register Patient**
2. **Update Medical Record**
3. **Access History via Telemedicine**
4. **Generate QR for Record Sharing**
5. **Share Records with Other Organizations**
6. **AI Emergency Alerts**
7. **Smart Reminders**
8. **Consent Dashboard**
9. **Manual Vitals Entry / Wearable Sync**

* **Relationships Between Actors and Use Cases**

|  |  |
| --- | --- |
| **Actor** | **Use Cases They Interact With** |
| **Patient** | Register Patient, Access History via Telemedicine, Generate QR, Smart Reminders, Consent Dashboard, Manual Vitals Entry, AI Emergency Alerts |
| **Physician** | Update Medical Record, Access History via Telemedicine, Share Records with Other Organizations |
| |  |  | | --- | --- | | **Receptionist** |  | | Register Patient |
| **External Organizations** | Generate QR for Record Sharing, Share Records with Other Organizations |
| **System (AI Logic)** | AI Emergency Alerts, Smart Reminders |

* Actor Role Table

|  |  |
| --- | --- |
| **Actor** | **Role description** |
| Patient | Main user; manages profile, records, reminders, and sharing permissions | |
| Physician | Views and updates medical records; conducts telemedicine sessions | |
| Receptionist | Assists with patient registration and scheduling | |
| External Org. | Receives shared records via QR or secure links | |
| System (AI Logic) | Monitors health data for emergencies; sends alerts and reminders | |

**Use case diagram**

Informal Use Case Descriptions

### 1. ****Register Patient****

A patient or receptionist opens the registration form on the dashboard. They enter personal details and medical history. The system checks for duplicates and securely stores the data in Firebase. If a duplicate is found, it alerts the admin or offers to merge records.

### 2. ****Update Medical Record****

A physician or nurse logs in and searches for a patient. They update the patient’s health data — maybe adding a diagnosis or treatment notes. The system saves the changes and timestamps them. If someone without permission tries to edit, the system blocks and logs the attempt.

### 3. ****Access History via Telemedicine****

The patient joins a virtual consultation using Jitsi Meet embedded in the browser. The physician accesses the patient’s records during the call. If the connection drops, the system offers to reschedule or switch to offline access.

### 4. ****Generate QR for Record Sharing****

The patient clicks “Generate QR” on their dashboard. A QR code appears, linking to their health records. An external provider scans it to view the data. If the code is expired or invalid, the system prompts the patient to regenerate it.

### 5. ****Share Records with Other Organizations****

The patient selects which records to share and chooses a destination (e.g., a lab or hospital). The system creates a secure link and sends it. If the link breaks or access is denied, the system alerts the patient.

### 6. ****AI Emergency Alerts****

The system scans patient records for critical health flags — like abnormal vitals or dangerous symptoms. If it detects an emergency, it sends an alert via email or dashboard. If it’s a false alarm, the patient can dismiss it and the system logs the event.

### 7. ****Smart Reminders****

The patient sets reminders for appointments, medication, or checkups. The system stores them and sends notifications via email or browser alerts. If a reminder is missed, it resends or escalates with a follow-up message.

### 8. ****Consent Dashboard****

The patient opens a dashboard showing who has access to their records. They toggle permissions for each organization. The system updates the access list and logs the changes. If someone tries to change permissions without authorization, the system blocks it.

### 9. ****Manual Vitals Entry / Wearable Sync****

The patient enters their vitals manually or syncs data from a wearable device. The system stores the metrics and uses them to trigger alerts or show trends. If the data is invalid, the system prompts correction.

Formal Use Cases

### 1. ****Register Patient****

* **Actor(s):** Patient, Receptionist
* **Preconditions:** Actor is authenticated and has access to the registration form
* **Main Flow:**
  1. Actor opens the registration form
  2. Enters patient demographic and medical history
  3. System checks for duplicate records
  4. System stores data in Firebase

 **Alternative Flow:**

* Duplicate detected → system prompts merge or alerts admin
* Invalid input → system prompts correction

 **Post conditions:** Patient profile is securely stored and accessible to authorized users

 **Technology Used:** Firebase Fire store, Firebase Auth

### 2. ****Update Medical Record****

* **Actor(s):** Physician, Nurse
* **Preconditions:** Actor is authenticated and authorized to edit records
* **Main Flow:**
  1. Actor logs in and searches for a patient
  2. Opens the patient’s record
  3. Edits and saves updates
  4. System timestamps the update
* **Alternative Flow:**
  1. Unauthorized access → system denies and logs attempt
  2. Network failure → system retries or alerts user
* **Post conditions:** Record is updated and securely stored
* **Technology Used:** Firebase Fire store

### 3. ****Access History via Telemedicine****

* **Actor(s):** Patient, Physician
* **Preconditions:** Both actors are authenticated and scheduled for a session
* **Main Flow:**
  1. Patient joins Jitsi Meet session via embedded link
  2. Physician accesses patient records during call
  3. Consultation is conducted
* **Alternative Flow:**
  1. Connection fails → system offers reschedule or offline access
* **Post conditions:** Consultation completed or rescheduled
* **Technology Used:** Jitsi Meet (iframe), Firebase Firestore

### 4. ****Generate QR for Record Sharing****

* **Actor(s):** Patient, External Provider
* **Preconditions:** Patient is logged in and has records to share
* **Main Flow:**
  1. Patient clicks “Generate QR”
  2. System creates QR code linked to records
  3. External provider scans QR to access data
* **Alternative Flow:**
  1. QR expired or invalid → system prompts regeneration
* **Post conditions:** External provider accesses records securely
* **Technology Used:** QRious.js or goqr.me API

### 5. ****Share Records with Other Organizations****

* **Actor(s):** Patient, External Organization
* **Preconditions:** Patient is authenticated and selects records to share
* **Main Flow:**
  1. Patient selects records and destination
  2. System generates secure link
  3. Link is sent to external organization
* **Alternative Flow:**
  1. Link broken or unauthorized → system alerts patient
* **Post conditions:** External organization accesses records
* **Technology Used:** Google Drive, Firebase Hosting

### 6. ****AI-Powered Emergency Alerts****

* **Actor(s):** System (AI Logic), Patient
* **Preconditions:** Patient records exist and system is actively monitoring
* **Main Flow:**
  1. System scans records for emergency flags
  2. If critical issue detected, sends alert via email or dashboard
* **Alternative Flow:**
  1. False alert → patient dismisses, system logs event
* **Post conditions:** Patient is notified or alert is dismissed
* **Technology Used:** JavaScript rule-based logic, EmailJS

### 7. ****Smart Reminders****

* **Actor(s):** Patient, System
* **Preconditions:** Patient is logged in and has a valid profile
* **Main Flow:**
  1. Patient navigates to “Reminders”
  2. Enters reminder details (date, time, type)
  3. System stores and schedules notification
  4. Notification is sent at scheduled time
* **Alternative Flow:**
  1. Reminder missed → system resends or escalates
  2. Invalid input → system prompts correction
* **Post conditions:** Reminder is logged and patient is notified
* **Technology Used:** Firebase Fire store, EmailJS, Browser Notifications

### 8. ****Consent Dashboard****

* **Actor(s):** Patient
* **Preconditions:** Patient is logged in and has shared data
* **Main Flow:**
  1. Patient opens dashboard
  2. Views current sharing permissions
  3. Toggles access for organizations
  4. System updates and logs changes
* **Alternative Flow:**
  1. Unauthorized toggle attempt → system denies and logs
* **Post conditions:** Permissions are updated and stored
* **Technology Used:** Firebase Fire store, HTML dashboard

### 9. ****Manual Vitals Entry / Wearable Sync****

* **Actor(s):** Patient
* **Preconditions:** Patient is logged in and has access to vitals form
* **Main Flow:**
  1. Patient enters vitals manually or syncs wearable
  2. System stores and analyzes data
* **Alternative Flow:**
  1. Invalid data → system prompts correction
* **Post conditions:** Metrics are saved and available for alerts or trends
* **Technology Used:** HTML form, optional Google Fit API