Sample Model

There is an AI model that looks at a patient's vital signs and lab numbers (age, blood pressure, heart rate, oxygen level, white blood cell count, diabatic) and answers one simple question:

"How likely is this patient to get worse soon (need urgent care, transfer to higher monitoring, or be readmitted)?"

It doesn't read notes or make diagnoses. It only looks at the numeric and yes/no facts you give it and returns:

- **A probability** (a number between 0 and 1) that quantifies the risk — e.g., 0.92 means "92% chance of deterioration according to the model."
- **A label** derived from that probability like low, medium, or high risk so humans or other systems can act.
- **A short explanation** listing the key features that pushed the prediction (e.g., "age, low oxygen, low blood pressure").

Example:

Input (what you send the model):

- Age: 72
- Blood pressure: low (85 systolic)
- Heart rate: high (110)
- Oxygen saturation: 88%
- WBC: 12.5 (elevated)
- Diabetes: yes
- Temp: 37.8

What the model returns:

- **Probability = 0.91** → "This person is very likely to deteriorate soon."
- **Label = high** → "Treat this as a high-priority case."
- **Explanation = age, low_spo2, hypotension** → "The biggest reasons are old age, low oxygen, and low blood pressure."

How to use the output

- If **label = high** and probability >= 0.80 → automatic escalation or clinician alert.
- If **probability is middling (0.60–0.80)** → flag for closer monitoring or human review.
- If **probability < 0.60** → routine care, no immediate escalation.

Below mentioned are the Test Cases to verify the above mentioned AI model.

ID: **TC001**
Title: Test case to check the correctness of the model with a single input.
Objective: To verify that the model returns expected response for a known input.
Type: Functional
Preconditions:

1. The model service is up and running.
2. Gold input with the expected response.

Test data: age=72, systolic_bp=85, heart_rate=110, spo2=88, temp=37.8, wbc=12.5, has_diabetes=1
Steps:

1. Send POST request to the model
2. Capture the response

Expected result: The response from the POST request should match with the expected response of gold input.
Pass/Fail: Pass if the response from the POST request matched with the expected response.
Severity: **Critical**

ID: **TC002**
Title: Test case to check the correctness of the model with multiple inputs.
Objective: To verify the model response for a set of known inputs.
Type: Functional
Preconditions:

1. The model service is up and running.
2. Excel file containing 50 gold inputs and their expected response.

Test data: 1-50 gold inputs from the excel file.
Steps:

1. Send POST request to the model for each input
2. Capture the response for each input.
3. Compare the response with the expected response

Expected result: The response from the POST request should match with the expected response of gold input.
Pass/Fail: Fail if the accuracy is less than 95%.
Severity: **Critical**

ID: **TC003**
Title: Test case to check the accuracy and confusion of the model.
Objective: To verify whether the model accuracy and confusion for 'High' and 'Medium' classes are with-in the thresholds or not.
Type: Performance
Preconditions:

1. The model service is up and running.
2. Excel file containing 200 inputs and their expected response.
3. Labeled test dataset with ground-truth (Label -> High/Medium/Low)

Test data: 1-200 gold inputs from the excel file.
Steps:

1. Send POST request to the model for each input
2. Capture the Label value from the response for each input.
3. Compute the confusion matrix for 'High' and 'Medium' classes.

Expected result: Accuracy should be greater 90% and Recall for 'High' should be greater than 0.95.
Pass/Fail: Fail if accuracy and Recall are below threshold.
Severity: **Critical**

ID: **TC004**
Title: Test case to check the correctness of the model in mapping the 'Label' from 'Probability'.
Objective: To verify Label mapping from Probability following the policy: High ≥ 0.8, Medium 0.6–0.8, Low < 0.6.
Type: Functional
Preconditions:

1. The model service is up and running.
2. Excel file containing 200 inputs and their expected response.
3. Inputs with known Probabilities.

Test data: Inputs with expected probabilities around boundaries (0.79, 0.80, 0.60, 0.59).
Steps:

1. Send POST request to the model for each input
2. Capture the Probability and Label value from the response for each input.

Expected result: The value of Label should be
High if Probability == 0.8; Medium if Probability == 0.79 or 0.60; Low if Probability == 0.59
Pass/Fail: Fail if any mapping violates the rule.
Severity: **High**

ID: **TC005**
Title: Test Case to check the behaviour of the model when any input value is missing.
Objective: To verify the how the model handles missing inputs (e.g missing spo2)
Type: Functional
Preconditions:

- The model service is up and running.
- All fields are mandatory.

Test data: Use the same input used for TC001 but omit the spo2 value.
Steps:

1. Send POST request to the model
2. Capture the response

Expected result: The system should return '400' response code.
Pass/Fail: Fail if response code is other than 400 and response body is present.
Severity: **Critical**

ID: **TC006**
Title: Test Case to check the behaviour of the model when any input value(s) is out of acceptable range.
Objective: To verify whether the model acts according to the document or not when it receives any input value that is out of acceptable range for the respective field. (e.g spo2 = 101, 150, -10)
Type: Functional
Preconditions:

1. The model service is up and running.

Test data: Three gold inputs with spo2 value as 101, 150 & -10 respectively
Steps:

1. Send POST request to the model
2. Capture the response

Expected result: The system should behave as mentioned in the documentation.
Pass/Fail: Pass if expected result matches with documented behaviour.
Severity: **High**

ID: **TC007**
Title: Test Case to check the behaviour of the model when it receives extreme but valid values for the input fields.
Objective: To verify whether the model remains stable or not on receiving extreme values for the input fields.
Type: Functional
Preconditions:

1. The model service is up and running.

Test data: Input with the age = 130, wbc = 100, spo2 = 10
Steps:

1. Send POST request to the model
2. Capture the response

Expected result: The response code: 200; Probability should be within 0-1
Pass/Fail: Fail if 5xx response code is received or missing response body
Severity: **High**

ID: **TC008**
Title: Test Case to check the consistency of the model when the same input is provided repeatedly.
Objective: To verify whether the model produces the same output or not when the same input is provided repeatedly.
Type: Functional
Preconditions:

1. The model service is up and running.

Test data: Same as test data used for TC001
Steps:

1. Send POST request to the model 15 times using the same input
2. Record the Probabilities.

Expected result: The model should respond with the same Probability all the 15 times.
Pass/Fail: Fail if the deviation is more than the threshold mentioned in the document.
Severity: **Medium**

ID: **TC009**
Title: Test Case to check the Latency & throughput of the model.
Objective: To verify the model performance as expected under the expected load.
Type: Performance
Preconditions:

1. The model service is up and running.
2. JMeter

Test data: Same as test data used for the test case TC002
Steps:

Run a simulated workload with 20 users, 100 req/s for 5 minutes and record p50/p95/p99.

Expected result: p95 should be less than 500ms and there should not be any errors.
Pass/Fail: Fail if p95 is >=500ms or error rate is > 1%
Severity: **High**

ID: **TC010**
Title: Test Case to check the Explanation field is feature-based with no PHI.
Objective: To verify whether the model is including any PHI in the Explanation response field or not.
Type: Security
Preconditions:

1. The model service is up and running.

Test data: Input with sample patient id in meta.
Steps:

1. Send a POST request to the model using the test data.
2. Capture the Explanation from the response body.

Expected result: Explanation should contain only feature names with no PHI
Pass/Fail: Fail if PHI is included in the Explanation
Severity: **Critical**

ID: **TC011**
Title: Test case to check the model's false-negative rate for high-risk cases.
Objective: To verify that the model does not miss too many high-risk cases.
Type:Safety
Preconditions:

1. The model service is up and running.

Test data: Inputs whose Probability is >= 0.8
Steps:

1. Send a POST request to the model using the test data.
2. Capture the Label from the response body.
3. Compute False Negative rate.

Expected result: FN rate should be less than the value mentioned in the document.
Pass/Fail: Fail if the FN rate is too high
Severity: **Critical**

ID: **TC012**
Title: Test case to check how model handles invalid/malformed input data.
Objective: To verify whether the model properly rejects or safely handles malformed inputs and return errors instead of crashing or producing incorrect predictions.
Type: Functional
Preconditions:

1. The model service is up and running.

Test data: Inputs with

- Missing request body
- Malformed request body
- age = "sixty-five"
- Spo2 = null
- Extra unexpected fields

Steps:

1. Send a POST request to the model for each input.
2. Capture response code, response body.

Expected result: The response code and response body should match with the documented result for the above inputs.
Pass/Fail: Pass if the model returns the documented results.
Severity: **Critical**

ID: **TC013**
Title: Test case to check how model handles injection & malicious input data.
Objective: To ensure the model service and backend are safe from injection attacks and malicious inputs.
Type: Security
Preconditions:

1. The model service is up and running.

Test data: Inputs with sql injection and malicious data
Steps:

1. Send a POST request to the model for each input.
2. Capture response code, response body and check for DB schema changes, record deletion, execution of shell commands.

Expected result: The response code and response body should match with the documented result for the above inputs and no DB schema changes, record deletion, execution of shell commands.
Pass/Fail: Fail if any DB mutation or system command occurs, stack traces returned, or sensitive data leaked.
Severity: **Critical**

UI/UX Test Cases

ID: **TC014**
Title: Test case to check the presence of all required input fields with proper labels.
Objective: To verify whether the input form is showing all the required fields with clear labels or not.
Type: Validation
Preconditions:

1. Input form page is loaded.

Test data: NA
Steps:

1. Inspect the input form
2. Verify each field mentioned in the document is visible with the correct label and placeholder.

Expected result: All the input fields with the correct label mentioned in the document should be present.
Pass/Fail: Fail if any required field is missing or mislabeled.
Severity: **High**

ID: **TC015**
Title: Test case to check the required fields are validated client-side.
Objective: To verify whether the user is able to submit the input form with missing critical data or not.
Type: Validation
Preconditions:

1. Input form page is loaded.

Test data: NA
Steps:

1. Enter values in all input fields except in spo2 field
2. Click the Submit button.

Expected result: Inline validation should be shown.
Pass/Fail: Fail if request is sent to the model server or if Inline validation is not shown for spo2 field.
Severity: **High**

ID: **TC016**
Title: Test case to check the min/max range validations for numeric fields.
Objective: To verify whether the min/max range validations are given for respective fields as mentioned in the document.
Type: Validation
Preconditions:

1. Input form page is loaded.

Test data: Inputs with spo2 = -5, spo2 = 125
Steps:

1. Try to submit the input form with above mentioned spo2 values.
2. Check the response after clicking on the Submit button.

Expected result: Inline validation for the spo2 field should be shown.
Pass/Fail: Fail if request is sent to the model server or if Inline validation is not shown.
Severity: **High**

ID: **TC017**
Title: Test case to check unit hints and conversion guidance are available for ambiguous inputs.
Objective: To verify whether the system shows unit hints or auto correcting the entered value or not.
Type: Functional
Preconditions:

1. Input form page is loaded.

Test data: Input with temp = 98.6 F
Steps:

1. Try to submit the form with the above input.

Expected result: The system should give an alert and offer conversion help.
Pass/Fail: Fail if no alert is given or if form is submitted.
Severity: **Medium**

ID: **TC018**
Title: Test case to check the handling of invalid inputs.
Objective: To verify whether the system validates and rejects invalid inputs (non-numeric, out-of-range, symbols, or empty values) or not.
Type: Functional
Preconditions:

1. Input form page is loaded.

Test data: Inputs with temp = abc, temp = @#!
Steps:

1. Try to submit the input form with the above temp values.

Expected result: Respective inline validation for the temp field should be shown.
Pass/Fail: Fail if no alert is given or if form is submitted.
Severity: **Medium**

ID: **TC019**
Title: Test case to check the input form is fully navigable via keyboard with the sensible focus order.
Objective: To verify whether the patient is able to navigate the form using the keyboard or not and the order of focus is logical or not.
Type: Usability
Preconditions:

1. Input page form is loaded with the focus set on the first field.

Test data: NA
Steps:

1. Use the keyboard to navigate from first to last field and submit.

Expected result: Focus moves in natural order and pressing Enter submits only when the focus is on the Submit button.
Pass/Fail: Fail on broken focus or skip of required fields.
Severity: **Medium**

ID: **TC020**
Title: Test case to check the form is usable and readable on mobile screen sizes.
Objective: To verify whether the input form is rendering correctly and remain usable on different mobile resolutions.
Type: Usability
Preconditions:

1. Input form page is loaded in mobile.

Test data: NA
Steps:

1. Resize the viewport to mobile and attempt to fill the form.

Expected result: No horizontal scroll, fields stacked, buttons accessible.
Pass/Fail: Fail on overflow or unreadable UI.
Severity: **Medium**

ID: **TC021**
Title: Test case to check help text/tooltips exist and explain ranges/units for input fields.
Objective: To verify whether key fields have text or tooltip explaining valid range and units or not.
Type: Usability
Preconditions:

1. Input form page is loaded.

Test data: NA
Steps:

1. Hover or click help icons for key fields.

Expected result: Tooltips show acceptable ranges, units, and examples.
Pass/Fail: Fail if text or tooltip is missing or incorrect.
Severity: **Low**

ID: **TC022**
Title: Test case to check client-side code doesn't log or leak PHI and network calls are secure.
Objective: To ensure the client side doesn't log or leak PHI in console logs.
Type: Security
Preconditions:

1. Input form page is loaded.

Test data: Same as test case TC010
Steps:

1. Fill form and submit.
2. Inspect browser console and network logs.

Expected result: No PHI printed to console or debug logs. Network requests should be over HTTPS and not include PHI in query strings.
Pass/Fail: Fail if PHI appears in logs or query parameters.
Severity: **Critical**