Appointment Booking

## Tool 1 : welcome\_message

### 1. Purpose

The welcome\_message tool is responsible for generating **personalized welcome messages** for patients based on their appointment history and current appointment status.

It helps in:

* Identifying whether the user is **new** (no appointments yet).
* Handling **existing users** with pending or confirmed appointments.
* Guiding patients who have **completed appointments** to their next steps.
* Presenting **actionable options** (e.g., schedule, reschedule, cancel, follow-up).

This ensures the patient always receives **context-aware messages** tailored to their situation.

### 2. Decorators Used

* @tool → Marks this function as a reusable tool (for orchestration pipelines, AI agents, or service integration).
* @dynamic\_unmask\_decorator → Ensures that patient-related sensitive information (like patient\_id) can be dynamically unmasked if needed for personalization while still handling data securely.

### 3. Function Definition

@tool

@dynamic\_unmask\_decorator

def welcome\_message(patient\_id):

"""

Generate welcome message for new user, existing user and appointment completed user.

Args:

patient\_id (str): ID of the patient.

Returns:

str: A personalized welcome message depending on appointment status.

"""

### 4. Input

* patient\_id **(str):**  
  Unique identifier of the patient.
  + Used to fetch the patient’s appointment history.
  + Helps in determining the user’s **appointment status context**.

### 5. Logic Flow

1. **Fetch Current Date**

current\_date = datetime.now().date()

1. **Retrieve Patient’s Closest Appointment**

nearby\_appointment = (

Appointment.objects.filter(patient\_id=patient\_id, status\_\_in=["completed","pending","reschedule\_pending","confirmed"])

.order\_by("-date", "start\_time").last()

)

* + Filters only relevant appointments (completed, pending, reschedule\_pending, confirmed).
  + Orders them in reverse chronological order (latest first).
  + Picks the **last appointment** (closest to the current date).

1. **Check Appointment Scenarios**
   * **Case 1: No appointment found**  
     → The user is considered **new**.  
     → Message suggests selecting reasons for a **new appointment** with symptom-based options.
   * **Case 2: Appointment Completed**  
     → The user recently finished an appointment.  
     → Message highlights the doctor, date, time, and status.  
     → Presents options:  
     @@Schedule appointment@@, @@Follow up@@, @@Upcoming appointment@@.
   * **Case 3: Appointment Pending / Reschedule Pending**  
     → The user has requested an appointment but it is **awaiting approval**.  
     → Message shows details with doctor, date, time, and status.  
     → Options:  
     @@Schedule appointment@@, @@Reschedule appointment@@, @@Cancel an appointment@@, @@Upcomming appointment@@.
   * **Case 4: Appointment Confirmed / Reschedule Confirmed**  
     → The user has an **approved appointment**.  
     → Message confirms doctor, date, and time.  
     → Options:  
     @@Schedule appointment@@, @@Reschedule appointment@@, @@Cancel an appointment@@, @@Upcomming appointment@@.

### 6. Example Responses

#### Case 1: New User (No Appointment)

hello [username]! I'm here to help to with your appoinment needs.,Please select a reason for your new appointment.

@@Fever@@ @@Cough@@ @@Back Pain@@ @@Headache@@ @@Stomach Pain@@ @@Skin Rash@@

@@Skip and check doctor availability@@

#### Case 2: Completed Appointment

Alert and show this to user, hello [username]! Your most recent appointment was with Dr. John on 2025-09-01 at 10:30 AM, and it has been completed. I'm here to help to with your appoinment needs

@@Schedule appointment@@ @@Follow up@@ @@Upcoming appointment@@

#### Case 3: Pending Appointment

Alert and show this to user, hello [username]! your nearest appointment on 2025-09-07 at 03:00 PM with dr. Smith status pending. Nurse will be approved your appointment request shortly. I'm here to help to with your appoinment needs

@@Schedule appointment@@ @@Reschedule appointment@@ @@Cancel an appointment@@ @@Upcomming appointment@@

#### Case 4: Confirmed Appointment

Alert and show this to user, hello [username]! Your nearest appointment on 2025-09-07 at 03:00 PM with dr. Smith status confirmed. I'm here to help to with your appoinment needs

@@Schedule appointment@@ @@Reschedule appointment@@ @@Cancel an appointment@@ @@Upcomming appointment@@

### 7. Key Notes

* Always wraps actionable options with @@ **prefix and suffix**.
* Does **not expose doctor’s specialization** when suggesting new appointment reasons.
* **Never uses bullet points or numbers** for listing options.
* **Follow-up appointment option is excluded** for new users (only valid for completed appointments).

### 8. Business Impact

* Provides a **personalized conversational experience**.
* Reduces user confusion by showing **context-specific next steps**.
* Standardizes how options are displayed across different appointment statuses.

## Tool 2 : followup\_appointment

### 1. Purpose

The followup\_appointment tool is responsible for **fetching the two most recent completed appointments** of a patient.

* Helps patients quickly **choose an appointment for a follow-up**.
* Ensures that the follow-up is always tied to a **recent, relevant appointment** (instead of old or unrelated visits).

### 2. Decorators Used

* @tool → Marks this as a reusable tool (exposed for orchestration pipelines or AI assistants).
* @dynamic\_unmask\_decorator → Dynamically handles unmasking of patient-sensitive identifiers like patient\_id while ensuring **data security**.

### 3. Function Definition

@tool

@dynamic\_unmask\_decorator

def followup\_appointment(patient\_id):

"""

Fetches the two most recent completed appointments for follow-up for a specific patient.

Args:

patient\_id (str): ID of the patient.

Returns:

List[json] or str: A list containing the details of the last two completed appointments,

or a message if none exist.

"""

### 4. Input

* patient\_id **(str):**
  + Unique identifier of the patient.
  + Used to fetch appointment records from the Appointment model.

### 5. Logic Flow

1. **Fetch Completed Appointments**

completed\_appointments = (

Appointment.objects.filter(patient\_id=patient\_id, status="completed")

.order\_by("-date", "-start\_time") [:2]

)

* + Filters all appointments where status = "completed".
  + Orders them in **reverse chronological order** (latest first).
  + Slices ([:2]) → Retrieves the **last two completed appointments only**.

1. **Check Availability**
   * If appointments exist → Format and return them.
   * If none exist → Return "No completed appointments available.".
2. **Formatting the Response**
   * Uses helper functions to ensure structured and readable data.
   * format\_completed\_appointments\_response()
     + Takes appointments list.
     + Converts each appointment into a dictionary (via format\_appointment).
     + Wraps them into a JSON string for UI/table rendering.
   * format\_appointment()
     + Extracts details like doctor’s name, date, time, reason, and status.
     + Converts raw DB fields into a **human-readable dictionary**.
   * convert\_to\_normal\_time()
     + Converts **24-hour time format** (HH:MM) → **12-hour AM/PM format** (hh:mm AM/PM).
     + Ensures appointment time ranges are easy for patients to understand.

### 6. Example Responses

#### Case 1: Two Completed Appointments Found

Here are the details of your latest completed appointments.

Please choose one of these appointments for your follow-up:

```AppointmentTable[

{

"id": 45,

"doctor\_name": "John",

"appointment\_date": "2025/08/25",

"time": "10:30 AM - 11:00 AM",

"reason": "Headache",

"status": "completed"

},

{

"id": 39,

"doctor\_name": "Smith",

"appointment\_date": "2025/08/10",

"time": "02:00 PM - 02:30 PM",

"reason": "Back Pain",

"status": "completed"

}

]```

#### Case 2: No Completed Appointments

No completed appointments available.

### 7. Key Notes

* Always returns **maximum 2 recent completed appointments**.
* Appointment details are **JSON-formatted** for easy integration with UI (tables/chatbots).
* Uses **helper functions** for clean separation of responsibilities:
  + format\_completed\_appointments\_response() → Presentation layer.
  + format\_appointment() → Data transformation.
  + convert\_to\_normal\_time() → Time formatting utility.
* Makes patient’s choice for **follow-up appointment clear and simple**.

### 8. Business Impact

* Enhances patient experience by showing **recent visit context**.
* Reduces errors by restricting follow-ups only to valid past appointments.
* Ensures **structured data output** that can be reused in different interfaces (UI, chatbots, APIs).

## Tool 3 : doctor\_availability\_calender

### 1. Purpose

The doctor\_availability\_calender tool is responsible for retrieving a **doctor’s availability calendar** for scheduling or rescheduling an appointment.

It:

* Fetches **PCP (Primary Care Physician) details** for a patient.
* Provides **availability slots** for either:
  + **Scheduling a new appointment**, or
  + **Rescheduling an existing appointment**.
* Supports **context-based interactions** (via thread\_id).

### 2. Decorators Used

* @tool → Marks this function as an AI agent tool, making it available for workflows or assistants.
* @dynamic\_unmask\_decorator → Dynamically handles unmasking of sensitive input values like patient\_id, doctor\_id, while maintaining data masking security if placeholders are used.

### 3. Function Definition

@tool

@dynamic\_unmask\_decorator

def doctor\_availability\_calender(

patient\_id: int,

doctor\_id: int,

reason: str,

appointment\_id: int = None,

thread\_id: int = None

):

"""

Purpose:

Retrieve PCP doctor details for a patient and provide availability calendar for rescheduling.

Args:

patient\_id (int): ID of the patient

doctor\_id (int): ID of the doctor

reason (str, optional): Reason for appointment

appointment\_id (int, optional): ID of the appointment for rescheduling

thread\_id (int, optional): ID of the thread for context

Returns:

str: PCP doctor or availability calendar details

"""

### 4. Inputs

* patient\_id **(int)** – Identifies the patient.
* doctor\_id **(int)** – Identifies the doctor to fetch availability.
* reason **(str, optional)** – Reason for the appointment (important for rescheduling context).
* appointment\_id **(int, optional)** – If provided, tool will handle **rescheduling flow**.
* thread\_id **(int, optional)** – Maintains conversational context (useful for chat/agent-based flows).

### 5. Logic Flow

1. **Masked Value Handling**

for param, value in locals().items():

if isinstance(value, str) and value.startswith("<<") and value.endswith(">>"):

print("masked\_value", value)

* + Detects if any parameter values are masked (<<value>>).
  + Ensures sensitive input is logged/handled carefully.

1. **Fetch Patient Details**

userdetail = UserAccount.objects.filter(id=patient\_id).first()

pcp\_doctors = userdetail.doctor

* + Retrieves the patient’s user account.
  + Gets assigned PCP doctors (not directly used, but indicates patient–doctor mapping).

1. **Rescheduling Flow (if** appointment\_id **is provided)**
   * Fetch appointment by ID.
   * If not found → return "Invalid Appointment ID".
   * Else → prepare doctor details from appointment record:
     + Doctor name, doctor\_id, hospital location, etc.
     + Mark type as "Rescheduling".
     + Specialization for referral appointment flow.
   * If reason provided → attach a thank-you note.
   * Return response with **calendar object embedded in JSON**:

```calendar{doctor\_json}```

1. **Scheduling Flow (if** appointment\_id **not provided)**
   * Fetch doctor’s hospital mapping:

doctor\_hospital = DoctorHospital.objects.filter(

hospital=userdetail.hospital,

doctor\_id=doctor\_id

).first()

* + Prepare doctor details with scheduling type.
  + Return response with calendar JSON:

```calendar{doctor\_json}```

### 6. Example Responses

#### Case 1: Scheduling a New Appointment

Here are the details of the patient's doctor. Please take a moment to select a slot from the doctor's availability calendar.

```calendar{

"doctor\_id": 23,

"doctor": "Dr. John Smith",

"location\_id": 5,

"location\_name": "Downtown Clinic",

"type": "Scheduling",

"specialization": ""

}```

#### Case 2: Rescheduling an Appointment (with Reason)

Thank you for providing the reason: 'Follow-up on blood test results' for rescheduling.

Please take a moment to select a slot from the doctor's availability calendar.

```calendar{

"doctor\_id": 23,

"doctor": "Dr. John Smith",

"location\_id": 5,

"location\_name": "Downtown Clinic",

"type": "Rescheduling",

"specialization": ""

}```

#### Case 3: Invalid Appointment ID

Invalid Appointment ID

### 7. Helper Function

def get\_appointment\_by\_id(appointment\_id):

return Appointment.objects.filter(id=appointment\_id)

* Fetches appointment records by ID.
* Used in the **rescheduling flow**.

### 8. Key Notes

* Returns structured output in the form of a **calendar JSON** (easy to render in UI).
* Clearly distinguishes between **scheduling** and **rescheduling** flows.
* Gracefully handles **invalid appointment IDs**.
* Includes **reason acknowledgment** if provided.
* Specialization is left blank ("") – It is used for referral appointment flow.

### 9. Business Impact

* Enables **patients to self-manage scheduling/rescheduling** seamlessly.
* Ensures **accurate doctor and location mapping** when offering slots.
* Provides **context-aware responses** (e.g., acknowledging rescheduling reasons).
* Improves patient engagement by making slot selection **interactive and structured**.

## Tool 4 : add\_new\_insurance

### 1. Purpose

The add\_new\_insurance tool is responsible for **adding insurance details** for a patient.

It ensures that:

* All required insurance information is provided.
* The insurance expiry date is valid (not already expired).
* Patient’s insurance record is updated in the system.
* Provides user-friendly success or error messages.

### 2. Decorators Used

* @tool → Marks the function as a reusable tool for integration with conversational agents, workflows, or APIs.
* @dynamic\_unmask\_decorator → Handles masking/unmasking of sensitive input values (e.g., insurance number, patient\_id).

### 3. Function Definition

@tool

@dynamic\_unmask\_decorator

def add\_new\_insurance(patient\_id, insurance\_number, insurance\_provider, insurance\_expiry\_date):

"""

Add new insurance details for a patient.

Args:

patient\_id (int): ID of the patient.

insurance\_number (str): Insurance policy number.

insurance\_provider (str): Provider of the insurance.

insurance\_expiry\_date (str): Expiry date of the insurance in YYYY-MM-DD format.

Returns:

str: Success or failure message.

"""

### 4. Inputs

* patient\_id **(int)** → Identifies the patient.
* insurance\_number **(str)** → Insurance policy number.
* insurance\_provider **(str)** → Name of the insurance provider.
* insurance\_expiry\_date **(str, YYYY-MM-DD)** → Insurance expiry date, must be in valid future date format.

### 5. Logic Flow

1. **Validation – Required Fields**

if not all([patient\_id, insurance\_number, insurance\_provider, insurance\_expiry\_date]):

return "Please provide all required insurance details"

* + Ensures that no field is missing.

1. **Validation – Expiry Date Check**

if datetime.strptime(insurance\_expiry\_date,"%Y-%m-%d").date() < datetime.now().date():

return "Insurance already expired. Please upload valid insurance"

* + Rejects expired insurance details.
  + Enforces correct date format (YYYY-MM-DD).

1. **Retrieve Last Insurance Record**

insurance\_rec = InsuranceDetails.objects.filter(user\_id=patient\_id).last()

* + Fetches the most recent insurance record for the patient.

1. **Update Insurance Details**

insurance\_rec.insurance\_number = insurance\_number

insurance\_rec.insurance\_expiry\_date = insurance\_expiry\_date

insurance\_rec.provider\_name = insurance\_provider

insurance\_rec.save()

* + Updates the existing record with new details.
  + Saves the changes back to the database.

1. **Return Success Response**

return f"Successfully added new insurance detail of Insurance provider: {insurance\_provider}, Insurance number: {insurance\_number} and Insurance Expiry date: {insurance\_expiry\_date} for the patient with ID {patient\_id}."

### 6. Example Responses

#### Case 1: Success

Successfully added new insurance detail of Insurance provider: LIC Health, Insurance number: ABC1234567 and Insurance Expiry date: 2026-08-15 for the patient with ID 42.

#### Case 2: Missing Fields

Please provide all required insurance details

#### Case 3: Expired Insurance

Insurance already expired. Please upload valid insurance

### 7. Key Notes

* **Expiry validation** is strict: insurance must always be valid at the time of entry.
* Currently **updates the last insurance record** for the patient — if no record exists, this might cause an error (NoneType).
* Does not handle **multiple active insurances** (assumes only one record per patient).
* Error handling for missing insurance\_rec is not implemented → can be enhanced.

### 8. Business Impact

* Ensures patients always have **up-to-date insurance information**.
* Prevents errors due to expired insurance being used.
* Supports **streamlined claims and billing** by keeping insurance records updated.
* Reduces manual verification needs for staff by validating at input stage.

## Tool 5 : get\_existing\_insurance\_details

### 1. Purpose

The get\_existing\_insurance\_details tool retrieves and formats active insurance details for a patient.

It supports:

* Displaying a patient’s valid insurance policies (not expired).
* Handling appointment-related context (pending, reschedule, complete pending, cancelled).
* Allowing patients to select insurance, change insurance, continue without insurance, or add new insurance.
* Providing role-based behavior (patient vs. staff).

### 2. Decorators Used

* @tool → Marks the function as a reusable tool.
* @dynamic\_unmask\_decorator → Handles sensitive data masking/unmasking for parameters like patient\_id, insurance\_number.

### 3. Function Definition

@tool

@dynamic\_unmask\_decorator

def get\_existing\_insurance\_details(patient\_id, thread\_id: int, appointment\_id: int = None, selected\_insurance\_id: int = None):

"""

Retrieve the existing insurance details for a given patient.

Args:

patient\_id (int): ID of the patient.

thread\_id (int): ID of the conversation thread.

appointment\_id (int, optional): ID of the appointment (used for rescheduling/confirmation context).

selected\_insurance\_id (int, optional): ID of a specific insurance selected by patient.

Returns:

str: JSON string containing formatted patient insurance details

along with actionable options like:

- @@Add new insurance@@

- @@Confirm proceed to schedule my appointment@@

- @@Schedule my appointment without insurance@@

"""

### 4. Inputs

* patient\_id (int) – Identifies the patient.
* thread\_id (int) – Thread context (conversation or episode).
* appointment\_id (int, optional) – Appointment reference for confirming/rescheduling.
* selected\_insurance\_id (int, optional) – When patient explicitly selects an insurance policy.

### 5. Logic Flow

1. Fetch Active Insurance Records

insurance = list(

InsuranceDetails.objects.filter(

user\_\_id=patient\_id,

insurance\_expiry\_date\_\_gte=datetime.now().date()

).values()

)

* + Retrieves only valid (non-expired) insurances for the patient.

1. Fetch Thread Context

thread = Episode.objects.filter(thread\_id=thread\_id).first()

1. Appointment Context (if provided)
   * Retrieves appointment.
   * Determines valid action options (confirm\_check) based on status:
     + pending or reschedule\_pending → @@Confirm appointment@@ @@Reschedule appointment@@ @@Cancel appointment@@
     + complete\_pending → @@Complete appointment@@ @@No Show@@
     + cancelled/cancelled\_waitlist → @@Show waitlist patient@@
2. If Insurance Exists
   * Build patient\_insurance\_details list with fields:
     + insurance\_id
     + insurance\_number
     + provider\_name
     + expiry\_date
     + insurance\_document (image reference)
   * Branching behavior:
     + Role = Staff (role\_id = 2)
       - If patient has no valid insurance → return "Currently no insurance available for patient …" + appointment actions.
       - Else → return "Insurance details for patient …" with insurance JSON and appointment actions.
     + If patient selected an insurance (selected\_insurance\_id)
       - Show selected insurance.
       - Return message with options:  
         @@Yes, Continue with this insurance@@ @@Change insurance details@@ @@Continue without insurance@@ @@Add new insurance@@
     + If multiple active insurances
       - Prompt user to pick one.
       - Return:  
         @@Add new insurance@@ @@Continue without insurance@@
     + If exactly one insurance exists
       - Return confirmation with options:  
         @@Continue with this insurance@@ @@Change insurance details@@ @@Continue without insurance@@ @@Add new insurance@@
3. If No Insurance Exists
   * Role = Staff (role\_id = 2) → "No Insurance active for patient …" + appointment actions.
   * Role = Patient → "No Insurance active for you @@Add new insurance@@"

### 6. Example Responses

#### Case 1: Multiple Active Insurances

These are your current Insurance. Please select the one you'd like to proceed with for scheduling this appointment,

or if you'd prefer, you can add a new insurance instead.

```InsuranceJSON[

{

"insurance\_id": 10,

"insurance\_number": "ABC123",

"provider\_name": "LIC Health",

"insurance\_expiry\_date": "2026-08-15",

"image": "insurance\_doc\_10.png"

},

{

"insurance\_id": 11,

"insurance\_number": "XYZ987",

"provider\_name": "HDFC Life",

"insurance\_expiry\_date": "2027-01-05",

"image": "insurance\_doc\_11.png"

}

]```

@@Add new insurance@@ @@Continue without insurance@@

#### Case 2: Single Active Insurance

Thank you for initiate appointment scheduling. Here is your current insurance.

You can proceed with scheduling the appointment or add new insurance if you prefer.

```InsuranceJSON[

{

"insurance\_id": 10,

"insurance\_number": "ABC123",

"provider\_name": "LIC Health",

"insurance\_expiry\_date": "2026-08-15",

"image": "insurance\_doc\_10.png"

}

]```

@@Continue with this insurance@@ @@Change insurance details@@ @@Continue without insurance@@ @@Add new insurance@@

#### Case 3: No Insurance Found (Patient)

No Insurance active for you @@Add new insurance@@

#### Case 4: No Insurance Found (Staff)

Currently no insurance available for patient John Doe

@@Reschedule appointment@@ @@Cancel appointment@@

### 7. Key Notes

* Uses polish\_message() from llmservice.nurse\_agent to reframe messages more naturally.
* Differentiates patient vs. staff workflow.
* Actionable options are always wrapped in @@.
* Insurance JSON is embedded in markdown code block InsuranceJSON… for easy frontend parsing.
* Handles multiple states: confirm appointment, reschedule, cancel, complete, no show, waitlist.

### 8. Business Impact

* Ensures patients always see valid insurance options.
* Provides flexibility to add/change insurance during appointment scheduling.
* Improves patient self-service while also assisting staff in managing appointments.
* Reduces chances of scheduling errors with expired insurance.

## Tool 6 : edit\_insurance\_details

### 1. Purpose

The edit\_insurance\_details tool is used to update an existing insurance record for a patient.

It allows updating:

* Insurance Number
* Insurance Provider
* Expiry Date

Additionally, it enforces validation for expiry dates and clears outdated insurance documents when updates are made.

### 2. Decorators Used

* @tool → Marks function as a reusable tool in the framework.
* @dynamic\_unmask\_decorator → Handles masking/unmasking of sensitive inputs like insurance\_number and insurance\_provider.

### 3. Function Definition

@tool

@dynamic\_unmask\_decorator

def edit\_insurance\_details(insurance\_id, insurance\_number=None, insurance\_provider=None, insurance\_expiry\_date=None):

"""

Edit existing insurance details for a specific insurance record.

Args:

insurance\_id (int): ID of the insurance record.

insurance\_number (str, optional): New insurance number to update.

insurance\_provider (str, optional): New provider name to update.

insurance\_expiry\_date (str, optional): New expiry date of the insurance in YYYY-MM-DD format.

Returns:

str: A message indicating success or failure of editing the insurance details.

"""

### 4. Inputs

* insurance\_id (int, required) – Identifies the insurance record to update.
* insurance\_number (str, optional) – New insurance number (cleans up $$ if passed).
* insurance\_provider (str, optional) – New provider name (cleans up $$ if passed).
* insurance\_expiry\_date (str, optional) – Must follow YYYY-MM-DD format. Validated to ensure it’s not already expired.

### 5. Logic Flow

1. Retrieve Insurance Record

insurance\_rec = InsuranceDetails.objects.filter(id=insurance\_id).first()

if not insurance\_rec:

return f"No insurance record found with ID {insurance\_id}."

* + If no record is found → returns failure message.

1. Update Fields (if provided)
   * Insurance Number

if insurance\_number:

insurance\_rec.insurance\_number = insurance\_number.replace("$$", "").strip()

* + Provider Name

if insurance\_provider:

insurance\_rec.provider\_name = insurance\_provider.replace("$$", "").strip()

* + Expiry Date

if insurance\_expiry\_date:

expiry\_date = datetime.strptime(insurance\_expiry\_date, "%Y-%m-%d").date()

if expiry\_date < datetime.now().date():

return "Insurance expiry date is expired. Please provide a valid expiry date. @@Insurance Expiry Date ---calendar---@@"

insurance\_rec.expiry\_date = expiry\_date

* + - Rejects expired dates.
    - Provides calendar input prompt (@@Insurance Expiry Date ---calendar---@@) when invalid.

1. Clear Insurance Document

insurance\_rec.insurance\_document = ""

* + Forces re-upload of insurance proof whenever details change.

1. Save Record

insurance\_rec.save()

return f"Successfully updated insurance details for insurance ID {insurance\_id}."

### 6. Example Responses

#### Case 1: Insurance Not Found

No insurance record found with ID 101.

#### Case 2: Expiry Date Already Passed

Insurance expiry date is expired. Please provide a valid expiry date. @@Insurance Expiry Date ---calendar---@@

#### Case 3: Successful Update

Successfully updated insurance details for insurance ID 45.

### 7. Key Notes

* Ensures data consistency by clearing outdated insurance documents whenever changes occur.
* Validates expiry date to avoid scheduling with invalid insurance.
* $$ cleanup ensures that masked/unmasked values from UI/chat inputs don’t get stored incorrectly.
* Returns user-friendly actionable prompts (@@...@@).

### 8. Business Impact

* Empowers patients or staff to easily update insurance details without creating duplicates.
* Prevents invalid/expired insurances from being stored.
* Ensures document compliance by requiring a re-upload on every edit.
* Supports smoother appointment scheduling with valid insurance data.