Prescription Builder – Functional Documentation

This document explains the important parts of the React-based Prescription Builder (App.js).

It is intended for clinicians (to understand what is captured and why) and developers (to maintain and extend the code).

The app builds an NDHM/NRCES-compliant FHIR R4 Document Bundle containing Composition, Patient, Practitioner, MedicationRequest(s), Condition, and optional Binary (PDF attachment).

Where you see comments like “//fetch via api”, those fields are placeholders meant to be supplied by backend or terminology services.

# High-level Workflow

1) Practitioner fills their details first → 2) selects/enters patient → 3) sets condition/diagnosis → 4) adds document (Composition) info → 5) adds one or more medications.

User hits “Generate JSON” to build a FHIR Document Bundle.

User can “Download JSON” or “Compare With Example” (modal showing a side-by-side diff with the reference file).

If a PDF is uploaded, it is embedded as Binary.data (base64).

# Utilities

## Function: uuidv4()

Purpose: Create a pseudo-random UUID for client-side resource IDs (sufficient for UI/demo).

Inputs: None.

Processing: Replaces x/y masks with random hex digits (RFC4122 v4 style approximation).

Outputs: A string like 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.

Used in: initializing medication rows and generating resource IDs inside buildBundle().

## Function: getISOWithOffsetFromDateInput(dateInput?)

Purpose: Formats dates into ISO 8601 with local timezone offset (e.g., 2025-08-12T14:35:21+05:30).

Inputs: Optional 'YYYY-MM-DD' from date inputs. If not provided, uses current date/time.

Processing: Combines provided date with current time, calculates timezone offset, replaces trailing 'Z' with '+/-HH:MM'.

Outputs: A dateTime string usable for FHIR 'instant' / 'dateTime' fields like meta.lastUpdated and Bundle.timestamp.

Note: The app also uses plain 'YYYY-MM-DD' for fields that require only a date (e.g., Patient.birthDate, authoredOn from date pickers).

## Const: pretty(o)

Purpose: Human-readable JSON output.

Inputs: Any object.

Processing: JSON.stringify with indentation.

Outputs: String used in preview and download.

## Function: lineDiff(leftStr, rightStr)

Purpose: Simple line-by-line comparator for the modal diff view.

Inputs: Two JSON strings (example vs generated).

Processing: Splits by newline, pairs lines, flags 'same' if lines match exactly.

Outputs: Array of rows consumed by the comparison modal.

# Application State (useState)

## State: practitioner

Fields: name, license.

Purpose: Who is issuing the prescription.

Validated: Both are mandatory before JSON generation.

## State: patient

Fields: name, mrn, birthDate, gender, phone.

Purpose: Patient demographic & identifier.

Validated: name, mrn, birthDate required.

## State: condition

Fields: text, code (SNOMED), clinicalStatus.

Purpose: Primary diagnosis (used in MedicationRequest.reasonCode and Condition resource).

Validated: text required.

## State: composition

Fields: title, status, date.

Purpose: Document-level (Composition) metadata shown before medications to match clinical flow.

Validated: title, status, date required.

## State: medications

Array of medication rows: medicationText, medicationCode (optional SNOMED), dosageText, authoredOn date, and optional administration details.

Purpose: Each entry becomes one MedicationRequest in the Bundle.

Key Logic: If medicationCode is blank, MedicationRequest.medicationCodeableConcept is text-only; if present, it becomes a coding[] block (with optional text).

## State: attachmentBase64

Purpose: Holds uploaded PDF (base64) to embed as Binary resource.

Optional: Present only if user attaches a PDF.

## State: generated

Holds the final Bundle object (preview + download).

## State: errorMsg

User-visible validation errors from buildBundle().

## State: diffRows

Rows for side-by-side diff in modal.

## State: showCompareModal

Boolean for toggling the comparison modal.

# Event Handlers and Core Logic

## Handler: handlePractitionerChange / handlePatientChange / handleConditionChange / handleCompositionChange

Purpose: Generic field updaters for their respective state objects.

Inputs: onChange event (name/value).

Outputs: Updated state.

## Handler: handleMedChange(index, field, value)

Purpose: Update a specific medication row.

Inputs: Row index, field name, new value.

Outputs: Updated medications array.

## Handler: addMedication()

Purpose: Add a new blank medication row with defaults.

Side Effects: Pushes a new object with a fresh UUID into medications.

## Handler: removeMedication(index)

Purpose: Remove a medication row; prevents deleting the last remaining row.

Edge Case: If only one medication exists, the button is disabled.

## Handler: handleFile(file)

Purpose: Accept only a PDF and store as base64 (Binary.data).

Validation: Rejects non-PDFs; reads file via FileReader; strips prefix to keep pure base64.

## Handler: handleGenerate(e)

Purpose: Validate inputs, call buildBundle(), and scroll to the JSON preview.

Errors: Displays validation errors via setErrorMsg.

## Handler: handleDownload()

Purpose: Download the generated JSON as a file.

Precondition: Requires 'generated' to exist.

## Handler: handleCompare()

Purpose: Compare current JSON with the sample (exampleBundle) in a modal.

Processing: pretty()-formats both, computes lineDiff().

# Core Builder: buildBundle()

Validates mandatory inputs (practitioner, patient, condition, composition, medications).

Creates new UUIDs for Composition, Patient, Practitioner, Condition, each MedicationRequest, and Binary (if any).

Bundle.meta: sets versionId and lastUpdated (dateTime with timezone).

Composition:

• resourceType: 'Composition'; profile set to PrescriptionRecord profile.

• subject: points to Patient (urn:uuid:...).

• author: points to Practitioner (urn:uuid:...).

• date: stored as full dateTime with timezone to preserve event timing.

• section[0].entry: references each MedicationRequest and optional Binary.

Patient:

• identifier uses MRN pattern and placeholder systems (to be replaced via API).

Practitioner:

• identifier set to license number with placeholder system (replace via API).

MedicationRequest (for each medication row):

• status: 'active', intent: 'order'.

• medicationCodeableConcept:

– If medicationCode is non-empty ⇒ coding[] with system=http://snomed.info/sct (plus optional text).

– If medicationCode is empty ⇒ only { text: medicationText } (to match reference example #2).

• authoredOn: uses the date input (YYYY-MM-DD) or composition date; keep as date-only per reference.

• reasonCode: mirrors Condition code/text; reasonReference links to the Condition resource.

• dosageInstruction: ALWAYS includes a single object with 'text'. Optional fields (additionalInstruction, timing, route, method) are included ONLY if user provided values (to match the example precisely when left blank).

Condition:

• clinicalStatus 'active' coding and SNOMED diagnosis code.

Binary (optional):

• contentType 'application/pdf'; data is base64 from the file picker.

Entry order in Bundle: Composition → Patient → Practitioner → MedicationRequest(s) → Condition → Binary.

# Clinician-facing: What you provide vs. what the system generates

You enter: Practitioner details, Patient details, Condition name/code, Composition title/status/date, and each Medication’s name, dose instructions, authoredOn date.

Optional: Additional instruction (e.g., 'With or after food'), timing (frequency/period/unit), route, method, and PDF attachment.

System generates: UUIDs, meta.versionId, meta.lastUpdated timestamps, some placeholder identifier systems, and Bundle identifiers (marked with “//fetch via api” in code).

Medication coding: If you provide a SNOMED code for the drug, it will be encoded in 'coding[]'; otherwise the medication is represented by plain text.

# Inferno / Conformance Considerations

Date shapes:

• 'authoredOn' is treated as a date-only (YYYY-MM-DD) per the reference example.

• Bundle.meta.lastUpdated and Composition.meta.lastUpdated are full dateTime with timezone.

DosageInstruction: leave only 'text' if you need a minimal surface identical to the reference; do not fill optional fields unless needed.

Profiles used: NRCES profiles for Bundle (DocumentBundle), Composition (PrescriptionRecord), Patient, Practitioner, MedicationRequest, Condition, Binary.

Terminology: SNOMED-CT codes are placeholders unless integrated with a terminology service.

# Developer Notes / Extension Points

Replace placeholder systems (e.g., http://hip.in, https://healthid.ndhm.gov.in) with real values from your environment.

Hook API calls where comments say “//fetch via api” (identifiers, SNOMED coding, practitioner registry, patient master, etc.).

Make sure your API returns SNOMED codes for medications when available to populate 'medicationCodeableConcept.coding'.

If you add e-signature support, consider adding a FHIR 'signature' element at Bundle level or digital signature in Composition per your compliance rules.

All dates should be captured via HTML date inputs and normalized using the helper functions.

# Quick Checklist

□ Practitioner name & license filled

□ Patient name, MRN, DOB filled

□ Condition text (and code if available)

□ Composition title, status, date

□ Each medication has a name and dosage text

□ Add SNOMED code for medication if available (optional)

□ Attach PDF if needed (optional)

□ Generate → Download or Compare