Quick Reference Guide

Developing a SMART on FHIR app

Contents

Topic: Introduction to SMART on FHIR and FHIR Basics

Fundamentals of FHIR: Resources, Profiles and APIs

Resources: FHIR uses a set of modular components called resources, which act as standardised building blocks for data. Each resource represents a specific concept or core data object (e.g. a Patient, Observation, Medication) and can be used independently or combined to support a wide range of clinical and administrative workflows.

Profiles: Profiles allow you to customise resource definitions by constraining which elements of the resource are required based on a particular use case.

They can be used to:

- Constrain a base resource e.g. making the Individual Healthcare Identifier (IHI) mandatory for the Patient resource
- Extend the base resource e.g. add information crucial for a specific context (such as Aboriginal or Torres Strait Islander status)
- Define conformance e.g. if an application or server is compliant with (for example) AU Core profile, interoperating systems will know what elements are expected and how to interpret them.

Application Programming Interface (API): An API is the facilitator that allows two distinct software to communicate securely and efficiently.

In the context of FHIR, APIs are almost always RESTful APIs (Representational State Transfer APIs), using the standard HTTP protocol's POST, GET, PUT and DELETE operations for create, read, update and delete (CRUD) interactions.

Overview of SMART on FHIR

What is SMART on FHIR?
While FHIR is the interoperability standard, SMART on FHIR is the security and integration layer that defines a set of technical arrangements for how applications access data within a Clinical System which is exposing data via FHIR. SMART on FHIR is the platform specification that standardises how third-party apps securely access FHIR data.

It connects user experience to infrastructure, authenticates the user, constrains what they can do, carries clinical context and makes apps reusable across organisations, turning standards compliance into scalable interoperability.

SMART on FHIR Architecture

the following are the 4 core components that make up SMART on FHIR architecture.

1. EHR System (Host Environment)
Description: Hosts or embeds SMART Apps.

Functions: Launch the Smart App in the context of the user's current patient and workflow. Optionally it can Host the SMART App.

2. FHIR Server (Data Layer)
Description: Stores and serves healthcare data in FHIR resource format (e.g., Patient, Observation, Encounter). A FHIR server can store data in non-native FHIR format and in a relational database. The exchange format must be in FHIR. The FHIR server acts as the resource server in OAuth 2.0 spec.

Note: in a real-world scenario the SMART on FHIR app is built against Clinical data residing in EHR/EMR and that component may host the data in a FHIR Server – so the EHR acts as FHIR Server.

Function: Acts as the API endpoint for SMART Apps to query or update data.

3. SMART Authorisation Server (Security Layer)
Description: Handles OAuth 2.0 and OpenID Connect protocols for secure access.

Function: Authenticates who is using the app (providing verifiable user identify details) and authorises what the app is allowed to do (scopes).

4. SMART App (Client Layer)
Description: A web or mobile app that requests authorisation and then interacts with the FHIR server using RESTful API calls.
Function: Can be launched within an EHR or operate as a standalone app.

Topic: SMART App Launch

Core Technical Concepts

OAuth 2.0 Authorisation Code Flow
This is the detailed protocol that facilitates the entire authorisation process, specifically the redirect and token exchange steps.

You will utilise OAuth 2.0 Authorisation Code Flow at the Discover Phase of the SMART App Launch.

Scopes
Scopes are the language used in the Authorisation Endpoint call to define what resources and what level of access (read, write, user, system) the app requests. They are the specific permissions that an OAuth access token grants to a client application to perform actions on behalf of a user.

You will come across Scopes at the Authorisation Request and Token Exchange phases.

Access Tokens & Refresh Tokens
These are the credentials provided by the Token Endpoint. The Access Token grants short-term access and the Refresh Token allows for non-interactive renewal.

Access Tokens are used at most stages of the process, while **Refresh Tokens** may be used at the Maintenance Phase of the SMART App Launch.

Launch Frameworks

The SMART on FHIR specification defines the secure, standardised methods by which an application initiates connection and obtains authorisation to access a FHIR server, typically hosted by an Electronic Health Record (EHR).
There are two user-facing EHR Launch and Standalone Launch patterns. The key difference between the two is where the launch is initiated and what clinical context is available to the application.

EHR Launch (Provider/Patient Context)

Initiation Point: Inside the EHR/Portal. User clicks a button/link within the EHR (e.g., Hospital Electronic Medical Record).
Authentication Flow: Context-first. The EHR user is usually already authenticated. The flow goes directly to authorisation and context injection.
Clinical Context: High Context. The EHR provides immediate,
pre-established context: the ID of the currently selected Patient (patient=...) and/or Encounter (encounter=...).
Launch Parameter: Requires the special launch parameter in the initial redirect, which is an

opaque handle to the context.
Primary Use Case: Workflow support: providing decision support, specialised charting, or a utility tool for a clinician actively using
the EHR.
Required Scope: Must include the launch scope to request the
in-EHR context.

Standalone Launch (Direct Access)

Initiation Point: Outside the EHR. User navigates directly to the app's URL (e.g., from a web browser, mobile app, or email link).
Authentication Flow: Authentication-first. The user must first select their organisation, then sign in with their EHR credentials
Clinical Context: Low Context. App starts with no context. User must typically select a patient after sign-in.
Launch Parameter: Does not require the launch parameter.
Primary Use Case: Patient Portals (PROMs apps), provider-facing dashboards, research data capture tools.
Required Scope: launch/patient is required to require user
to select the patient as part of the authorisation process.
Requests patient/user access scopes directly (e.g., patient/*.rs).

App Registration and Configuration

Registering Your App with a SMART Authorisation Server
Once you have identified your launch pattern, you will need to register your SMART on FHIR app with a SMART Authorisation Server. Current Landscape, traditional SMART on FHIR app registration is an emerging capability in Australia. The availability varies significantly by context.

Current Availability:

- My Health Record FHIR Gateway - Consumer-facing apps can register through the Digital Health Developer Portal. Note: This supports patient access via MyGov, not EHR-embedded clinician workflows.

- Provider Connect Australia - SMART on FHIR ready for provider directory data.
- Some international EHR deployments - Epic and Cerner may support SMART in some Australian hospital implementations (verify directly with the hospital's EHR team).

In Development: Australian PMS vendors (MedicalDirector, Best Practice) are implementing FHIR capabilities. Contact vendors directly for current SMART on FHIR app registration support.

The registration process below describes the typical steps when SMART on FHIR is available. Always verify current capabilities with your target platform.

The typical steps for getting started are:

Step 1
Identify which EHR you need to connect to.

Step 2
Contact the Vendor’s development programme and register to join their developer community.

Step 3
Submit required information, which typically includes:
• App Name and Authorisation redirect URI(s)' (for OAuth 2.0)
• App's requested scopes (e.g., launch patient/Observation.rs)
• If using EHR Launch pattern, App Launch URL
• Evidence of your app's security and privacy compliance.

Step 4
To connect to some services (e.g. MyHealth Record) you may need to obtain appropriate test certificates (e.g. NASH PKI test certificate).

Step 5
Go live approval (vendor will need to be satisfied with your app functionality and compliance for connection in their production environments).

Resources

- Digital Health Developer Portal
- Healthcare Identifiers (HI) Service
- My Health Record Portal
- NASH
- PCA (Provider Connect Australia) - provider directory data.
- PRODA (Provider Digital Access) - registration required for developers building commercial or third-party software for the Australian digital health services (like PCA).

Topic: FHIR Resources and APIs for Developers

FHIR Data Formats and Context

The role of JSON and XML for data exchange in APIs

JSON (JavaScript Object Notation) and XML (Extensible Markup Language) are alternative formats for exchanging data in APIs. Both are human readable, text-based formats that are used for storing and exchanging data.

JSON is the preferred format for modern web applications due to its simplicity, compact structure and faster parsing. XML remains valuable when applications require complex data structuring and robust schema validation.

Feature	JSON	XML
Structure	Key-value pairs ("key": "value") and arrays ([]). Minimal structural characters (curly braces, colons, commas).	Tag-based structure, requiring both opening (<tag>) and closing (</tag>) tags for elements.
Verbosity	**Low.** It is syntactically lightweight due to less overhead.	**High.** It is generally more word-heavy due to mandatory closing tags and often includes verbose attribute definitions.
Parsing Speed	**Faster.** Native compatibility with JavaScript engines leads to quicker processing and less overhead for web applications.	**Slower.** Requires more complex parsing logic to handle the opening/closing tag structure and document hierarchy.

FHIR API Interaction: Basic CRUD

Introduction to RESTful FHIR

FHIR uses RESTful APIs, meaning it follows REST (Representational State Transfer) architectural principles where operations on resources are performed using standard HTTP methods.

When developing FHIR applications, you need to understand how FHIR operations map to HTTP verbs. Together, these actions (Create, Read, Update and Delete) are represented by the acronym “CRUD”. In FHIR-based systems, 'Search' is an additional critical operation used extensively to query and retrieve data.

When developing FHIR-compliant applications, it's essential to understand these FHIR operations first and then how each one maps to the underlying HTTP verbs used to interact with health data.

CRUD Action	HTTP Verb	FHIR Purpose	Example Scenario	Construction Example
Create	POST	Submit a new resource to the server. Server assigns the resource ID.	Creating a new MedicationRequest.	Request format targets the resource type URL (the resource collection URL). POST [FHIR Base URL]/MedicationRequest HTTP/1.1
Read	GET	Retrieve a resource by its ID or search for a collection of resources.	Retrieving a specific Patient record or all Observations.	To instruct the server to retrieve a specific, single data packet you would use the following format: [FHIR Base URL]/[ResourceType]/[ID] Together with a Patient resource id (e.g. 123), the request URI would be: GET https://fhir.server.com/Patient/123
Update	PUT	Replace or update an existing resource at a specific, known URI.	Changing the status of an Encounter from 'active' to 'finished'.	Format is defined by the HTTP request structure, which requires a specific URL and a data message (JSON or XML) containing the full, updated resource. PUT [FHIR Base URL]/Patient/example-id HTTP/1.1
Delete	DELETE	Remove a specific resource from the server (often restricted in clinical systems).	Removing a resource e.g. temporary draft DocumentReference.	The request targets the specific resource instance for deletion by including its Resource Type and its ID. DELETE [FHIR Base URL]/[ResourceType]/[ID]
Search	GET	Find resources that match specific criteria using FHIR search parameters. Returns a Bundle containing zero or	Finding all Observation resources for a patient (e.g. patient ID 123) within a certain category or date range.	To search a collection of resources, send a GET request to the resource type URL with search parameters in the query string. Example – search for all Observations for patient 123: GET https://fhir.server.com/Observation?patient=123

		more matching resources. Basic searches use GET with query parameters on the resource type (collection) endpoint.		FHIR also allows POST-based search using: POST [FHIR Base URL]/Observation/_search (with parameters in the body).

FHIR Search Techniques: from Basic to Advanced

In real-world healthcare applications, you rarely want all resources of a type - you need specific data. A clinician doesn't want every patient in the system; they want their patients. A pharmacy doesn't need all prescriptions; they need pending ones for today. FHIR's search capabilities let you retrieve exactly the data you need, efficiently.
Basic FHIR Search: FHIR search uses standard HTTP GET requests with query parameters to filter resources. This follows RESTful API patterns you already know. See the example below.

Format: GET [FHIR BASE URL]/[Resource Type]?[Parameter]=[Value]
Example: Find all patients with family name "Chen"
Code: GET /Patient?family=Chen

AU Core Search Parameters
FHIR provides standard search parameters (like name, birthdate, identifier) and AU Core adds Australian-specific parameters for local healthcare contexts.

Key AU Core Search Parameters:

Parameter	Related Resources	Use Case
discharge-disposition	Encounter	Filter hospital encounters by how the patient was discharged (e.g., home, transferred, deceased)
indigenous-status	Patient	Search and report on Aboriginal and Torres Strait Islander status for health equity monitoring
gender-identity	Patient, RelatedPerson, Person, Practitioner, PractitionerRole	Search by gender identity (distinct from biological sex)
supporting-info	ServiceRequest	Find service requests with specific supporting documentation or references

Comparison: _include, _revinclude, Chaining, _include:iterate

Feature	_include	_revinclude	_include:iterate	Chaining
Purpose	Retrieve related	Retrieve related	Multi-hop expansion	Filter by

	resources (forward)	resources (backward)		relationship
What it does	Follows references you make	Finds what references you	Chains across multiple levels	Filters by related criteria
What you get	Primary + referenced resources	Primary + referencing resources	Primary + multiple levels of resources	Primary only (filtered)
Example result	Observations + Practitioners	Patient + their Observations	Obs + Pract + Organisations	Encounters (filtered)
Use when	Need what you point to	Need what points to you	Need multi-level relationships	Filtering by related properties

Topic: App Development Foundations

Introduction to SMART on FHIR client libraries.

What is a SMART on FHIR client library?
A SMART client library is pre-written code that handles the complex tasks of building a SMART on FHIR app.

Instead of manually coding OAuth authorisation flows or FHIR resource operations from scratch, you use a library that provides ready-made functions.

For example, instead of writing hundreds of lines of code to implement SMART Launch, you call a single library function that does it for you.

SMART on FHIR client libraries handle two critical tasks: FHIR operations (reading, creating, searching resources) and SMART authorisation (OAuth 2.0 flows). This means you can focus on building your app's features rather than implementing security protocols.
SMART on FHIR client libraries by Language:

Language	Library	Best For
JavaScript	fhirclient	Web apps (React, Angular, Vue)
Python	fhirclient-py	Backend services, automation
Java	HAPI FHIR	Enterprise applications
C#	Firely .NET SDK	.NET applications

These libraries implement the OAuth 2.0 authorisation patterns you learned earlier (SMART Launch and Backend Services) so you don't have to write OAuth 2.0 code manually.

Using a SMART Sandbox

A SMART sandbox is a safe testing environment that simulates a real FHIR server and EHR system.

It allows you to develop and test your SMART app without accessing real patient data or affecting production systems.

You can make mistakes, test features and learn how OAuth 2.0 flows work, all without any risk.

Registration connects your app to the sandbox and provides the credentials you need for testing:

What you provide:
- Your app's name and description
- Where your app runs (URLs)
- What data your app needs (scopes)

What you receive:
- Client ID (unique identifier for your app)
- OAuth endpoints (where to send authorisation requests)
- Test patient data to work with

This registration process mirrors what you'll do when deploying to a real EHR system, but in a safe environment where you can experiment and learn.

Popular SMART Sandboxes

SMART Health IT App Launcher
- Free, public sandbox
- International standard
- Pre-loaded test patients
- https://launch.smarthealthit.org

My Health Record FHIR Gateway (Australian Context)
- Australian government sandbox
- AU Core compliant
- Supports Australian identifiers and profiles
- For detailed setup guide, see: My Health Record FHIR® Mobile Gateway resource

Note: Registration steps vary slightly between sandboxes, but the core concept is the same - provide your app details, receive credentials, test your OAuth flows.

Framework resources

Choosing a User Interface (UI) Framework

SMART on FHIR and the fhirclient library work with any modern JavaScript framework.

This means you can choose the UI technology that best fits your project needs and team experience.

Angular, React and Vue.js are the three most popular frameworks for building web-based SMART apps. They all work well with SMART on FHIR - the choice depends on your specific context.

Framework Comparison

Framework	Best For	Learning Curve	Key Strength
Vue.js	Beginners, small projects, rapid prototyping	Easiest	Simple, gentle learning curve, low complexity
React	Flexible projects, large community support	Moderate	Most popular, extensive resources, flexible
Angular	Enterprise apps, large teams, structured projects	Steepest	Full-featured, opinionated structure, TypeScript

Topic: Read and Write Workflows with FHIR Services

Handling Responses and Servers

When working with FHIR servers, you'll encounter two important concepts: Bundles (for handling multiple resources) and OperationOutcomes (for understanding errors).

Let's look at Bundles first. A Bundle is a container that groups multiple related FHIR resources together. Bundles are used throughout FHIR for different purposes.

Historical Versions
Return multiple versions of a resource showing changes over time.
Example: View all previous versions of a patient's medication list.

Search Results
Results from search queries are returned in a Bundle.
Example: Searching for "all observations for Patient 123" returns a Bundle containing all matching Observation resources.

Transaction/Batch Operations
Execute multiple create, update, or delete operations in one request.

Example: Create a new patient and their initial observations in a single API call - all operations succeed or all fail together.

Clinical Documents
Group resources that form a complete clinical document.
Example: A discharge summary Bundle containing the patient, encounter, medications and care plan.

Message Exchange
Send related resources as part of a healthcare message.
Example: Transferring a patient's complete record from one system to another.

Why Bundles matter: They reduce network requests (one call instead of many) and ensure transactional integrity for batch operations. For more information, see the FHIR Bundle specification.

Topic: App Design, Testing and Deployment

SMART scopes

SMART scopes control what data your app can access.

SMART scopes are introduced here in planning because your scope requirements directly influence your app's security architecture and which FHIR resources you'll use.

Following the principle of least privilege, request only the minimum scopes necessary to accomplish your use case.

The technical OAuth 2.0 implementation is covered later in this topic.

Scope Format: [context]/[resource].[permission]
Example SMART scopes for a diabetic risk app:

- patient/Patient.rs - Read patient demographics
- patient/Observation.rs - Read lab results
- patient/Condition.rs - Read diagnoses

Establish parameters that request only the minimum necessary scopes to protect patient privacy.
Bad: patient/.cruds (access to ALL patient data)
Good: patient/Observation.rs patient/Patient.rs (specific resources only).

Human-Centred Design in Health IT

Human Centred Design (HCD) is a framework that puts people (the app users) at the heart of the

design and delivery process.

HCD is comprised of a range of approaches that can help you to understand the end-users' needs, behaviours, workflows and pain points.

By approaching your solution development in this way, you can create a user experience that meets their needs, streamlines their workload and increases the adoption rate and success of the application.

Considerations when you are designing your application

Solve the root issue - Define the core problem the application will solve. Focus on the root causes rather than just the symptoms.

Focus on people - Understand the user's context, motivations and pain points through research (e.g., interviews, observations etc.).

Understand the process - Recognise that every part of the application is a system with interconnected parts.

Iterate and test - Continuously test prototypes with real users to gather feedback and improve the application.

Deployment and Monitoring

Continuously test prototypes with real users to gather feedback and improve the application.

Deploying a SMART app in Australia typically involves integrating with the My Health Record system through the Australian Digital Health Agency (The Agency).

My Health Record Integration Process Complete these nine steps to gain production access:

Step 1: Register with Developer Community
Create an account at https://developer.digitalhealth.gov.au/user/register(opens in a new tab)

You receive: Access to developer documentation and requirements.

Step 2: Review Key Documents
Review materials at
https://developer.digitalhealth.gov.au/resources/guides(opens in a new tab)

Key documents: App Vendor Guide, API specifications, security requirements, conformance profiles.

Step 3: Obtain Test Certificate (If Required)
Required for Interaction Model 4 only (server-to-server communication)
Result: Test certificate for development environment.

Step 4: Register Your Software
Complete the Portal Operator Registration Form with organisation and app details.

You receive: Test kit including test cases, test data and credentials for test environment.

Step 5: Development and Testing
Develop your app using the test environment, API specifications and provided test data.
Resources: API documentation, FHIR implementation guide, technical support from the Agency.

Step 6: Self-Assessment Testing
Test your app using provided test cases and document results.

Contact for guidance:
• Email: help@digitalhealth.gov.au(opens in a new tab)
• Phone: 1300 901 001
Test areas: All API endpoints, authentication flows, consent management, error handling and data write-back.

Result: Complete test history ready for formal review.

Step 7: Formal Notice of Connection (NOC) Testing
Participate in observed virtual testing session with National Infrastructure Operator (NIO).

Process: Demonstrate app functionality while NIO observes and verifies test cases against API specifications.

Result: Declaration of Notice of Connection issued upon successful completion.

Note: This is the formal gateway to production access.

Step 8: Obtain Production Certificate (If Required)
Obtain a certificate for the intermediary server (click here(opens in a new tab) for more information).

Outcome: Certificate issued.

Step 9: Production Access Authorisation
Submit required documentation:

a) Production Environment Access Request (PEAR) form
Complete technical details and confirm conformance.

b) Portal Operator Registration Agreement (PORA)
Legal agreement with ADHA outlining terms and obligations.
Result: PRODUCTION ACCESS GRANTED.

You become a Registered Portal Operator under the My Health Records Act 2012.

Deployment and monitoring checklist

Pre-Deployment
☐ My Health Record integration
☐ Received Notice of Connection
☐ Obtained production credentials
☐ Configured monitoring and analytics
☐ Prepared incident response plan

Post-Deployment
☐ Monitor technical performance continuously
☐ Collect and analyse user feedback
☐ Maintain prioritised improvement backlog
☐ Release updates regularly