National Institutes of Health
Office of Data Science Strategy

SMART on FHIR® Workshop

National Institutes of Health (NIH), Office of the Director (OD), Office of Data Science Strategy (ODSS)

June 22, 2023

FHIR® is the registered trademark of Health Level Seven International (HL7). Use of the FHIR trademark does not constitute an HL7 endorsement of this workshop.

Agenda

1. Introduction slides
2. SMART App Launch workflow + sample app walk-through
3. FHIR API introduction
4. US Core introduction
5. FHIRPath introduction
6. Hands-on: make your own SMART on FHIR app

Introduction

What is SMART on FHIR?

> The goal of the original SMART on FHIR API is audacious and can be expressed concisely: an innovative app developer can write an app once and expect that it will run anywhere in the health care system.
>
> …
>
> SMART provides a full stack of open specifications that enable a medical apps platform.

— https://smarthealthit.org/smart-on-fhir-api/

What is SMART on FHIR?

- SMART stands for “Substitutable Medical Apps, Reusable Technology”, a standard by the SMART Health IT group
- Based on open standards: FHIR, OAuth2, OpenID

  Connect Widely implemented/used (e.g., on all iPhones)
- Required as part of ONC certification for 21st Century Cures Act: §170.315(g)(10) Standardized API for patient and population services
  - § 170.215(a)(4) lists the specific APIs including SMART Application Launch

How can SMART on FHIR help research?

SMART on FHIR lets you:

- Integrate with an EHR (e.g., add an AI-driven CDS app) Add patient-generated data to an EHR workflow
- Create an app that can be used acorss institutions and EHR products
- Access Bulk Data

SMART on FHIR standards

It ties together existing common web standards and HL7 specifications to enable secure EHR integration:

- OAuth2 for authorizing a third-party app
- OpenID Connect for authenticating a patient or provider HL7 FHIR for data modeling and API
- JSON for the data format
- HL7.FHIR.UV.SMART-APP-LAUNCH standard for
- launching from EHR
- HL7 CDS Hooks for triggering based on EHR actions (see next slide)

Aside: CDS Hooks

- CDS Hooks is an HL7 standard that can support SMART on FHIR application integration with EHRs
- They allow an action in an EHR to trigger an action in a third-party application
- For example, a patient-view hook is triggered when the patient record is opened, which could then call natural language processing software

Technical considerations

SMART apps have multiple authorization patterns

- SMART App Launch
  - EHR Launch: user launches an application from within an EHR (ex: a CDS app)
  - Standalone Launch: user launches the application directly (ex: iPhone Health app)
- SMART Backend Service: support applications that run autonomously (ex: data pipeline)

Technical considerations, continued

Security

- Use reputable open-source software libraries to save development time and avoid common security pitfalls. SMART Health IT lists SMART-on-FHIR software libraries.

Privacy

- FHIR servers will likely return sensitive healthcare data. PHI rules will likely apply. You must also comply with your institution’s IRB and privacy rules.

Technical considerations, continued

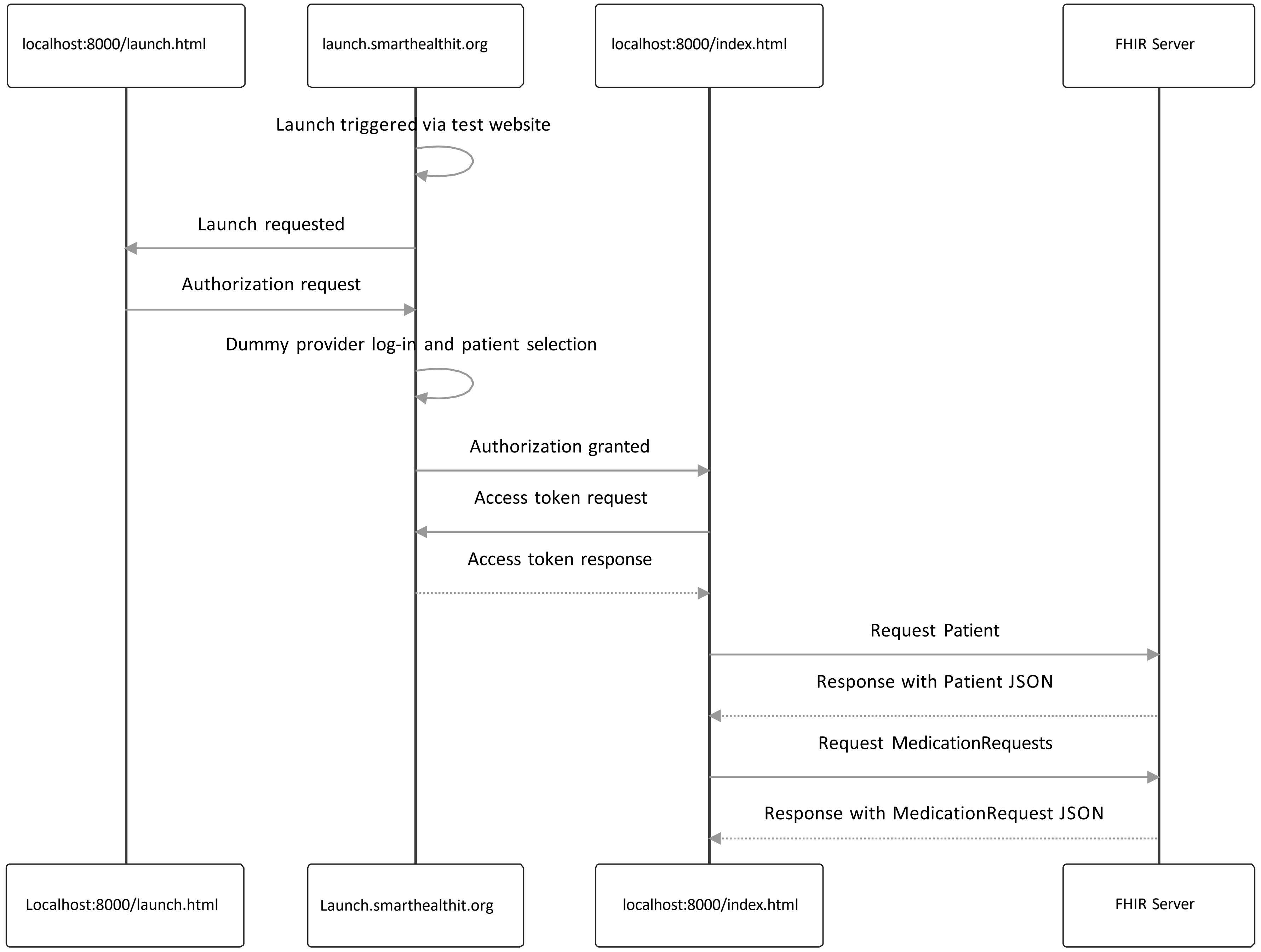
Design

- If you are developing a user-facing application, consider a human-centered design approach to help ensure good user experience.
- More: https://www.fastcompany.com/90772846/human- centered-design

SMART App Launch workflow + sample app demo

SMART App Launch workflow

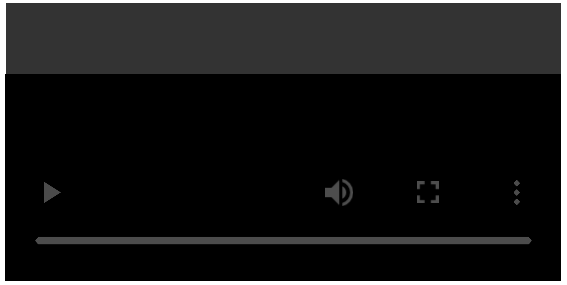
- Described in detail in the SMART App Launch spec Diagram specific to this workshop on next slide
- We will be using https://launch.smarthealthit.org to simulate EHR launch
  - Some differences vs. production implementation


localhost:8000/launch.html
launch.smarthealthit.org
localhost:8000/index.html
FHIR Server
Launch triggered via test website
Launch requested
Authorization request
Dummy provider log-in and patient selection
Authorization granted
Access token request
Access token response
Request Patient
Response with Patient JSON
Request MedicationRequests
Response with MedicationRequest JSON
Localhost:8000/launch.html
Launch.smarthealthit.org
localhost:8000/index.html
FHIR Server

Basic sample app

- Source: https://purl.org/fhir-for-research/workshops/smart-on-fhir/sample-app-git
- Explanation: https://purl.org/fhir-for-research/web/modules/smart-on-fhir-tech
- Options for accessing sample app:
  - Should be running on http://localhost:3000 for you
- Also available at https://purl.org/fhir-for-research/workshops/smart-on-fhir/sample-app-build Go to https://launch.smarthealthit.org
  - Enter URL for sample app’s launch.html page into “App’s Launch URL” and click “Launch”

Video of launch sample app via SMART App Launch



Link to video:

https://purl.org/fhir-for- research/workshops/smart-on-fhir/flow-video

Basic sample app structure

```
smart-on-fhir-demo/
          |
          +--- launch.html
          |
          +--- index.html
```

You can copy file contents from https://purl.org/fhir-for-research/workshops/smart-on-fhir/sample-app-git.

If you cloned the repository this is already done.

SMART on FHIR walkthrough

Step 1. *launch.smarthealthit.org* simulates an EHR and triggers a SMART App Launch

Your browser (a client) gets redirected by the EHR to your app's http://localhost:3000/launch.html with the following parameters:

- iss: Identifies the EHR's endpoint for the app
- launch: An opaque identifier for this specific app launch and EHR context, required for security purposes
    - JavaScript librar automatically passes this back to EHR with authorization request (Step 2)

SMART on FHIR walkthrough

Step 2. Your app’s launch.html executes an authorization request with select parameters

- We are using the SMART on FHIR JavaScript Library from SMART Health IT
- This library handles the OAuth2 workflow and making authenticated requests from the FHIR server

SMART on FHIR walkthrough

Step 2. Your app's launch.html executes an authorization request with select parameters

```
<script>
    FHIR.oauth2.authorize({

      // The client_id that you should have obtained after registering a cl
      // the EHR.
      //
      // Note that this can be an arbitrary string when testing with
      // http://launch.smarthealthit.org.
      clientId: "my_web_app",

      // The scopes that you request from the EHR. In this case we want to:
      // launch            - Get the launch context
      // openid & fhirUser - Get the current user
      // patient/*.read    - Read patient data
      scope: "launch openid fhirUser patient/*.read",

       // (where the launchUri is), you can omit this option because the def
      // " " However some servers do not support directory indexes so " "
```

SMART on FHIR walkthrough

The clientId parameter is a specific string obtained after registering the app in the EHR manually. Replace "my_web_app" with your specific app identifier.

```
<script>
    FHIR.oauth2.authorize({

      // The client_id that you should have obtained after registering a cl
      // the EHR.
      //
      // Note that this can be an arbitrary string when testing with
      // http://launch.smarthealthit.org.
      clientId: "my_web_app",

      // The scopes that you request from the EHR. In this case we want  to:
      // launch            - Get the launch context
      // openid & fhirUser - Get the current user
      // patient/*.read    - Read patient data

      scope: "launch openid fhirUser patient/*.read",
       // Typically, if your redirectUri points to the root of the current d
      // (where the launchUri is), you can omit this option because the def

```

SMART on FHIR walkthrough

The scope parameter specifies what kinds of data the app needs access to. See SMART on FHIR scope and lunch context for more data access options.

```
        // the EHR.
        //
        // Note that this can be an arbitrary string when testing with
        // http://launch.smarthealthit.org.

        clientId: "my_web_app",

        // The scopes that you request from the EHR. In this case          we want   to:
        // launch                - Get the launch context
        // openid & fhirUser - Get the current user
        // patient/*.read       - Read patient data
        scope: "launch openid fhirUser patient/*.read",

        // Typically, if your redirectUri points to the root of the current  d
        // (where the launchUri is), you can omit this option because the  def
        // ".". However, some servers do not support directory indexes so "."
        // will not automatically map to the "index.html" file in that  direct

    });
```

SMART on FHIR walkthrough

redirectUri is where the EHR will redirect the web browser (client) to after authorization. In this case it is the app’s index.html.

```
    //
    // Note that this can be an arbitrary string when testing     with
    // http://launch.smarthealthit.org.
    clientId: "my_web_app",

    // The scopes that you request from the EHR. In this case     we want  to:
    // launch                 - Get the launch context
    // openid & fhirUser - Get the current user
    // patient/*.read         - Read patient data
    scope: "launch openid fhirUser patient/*.read",

    // Typically, if your redirectUri points to the root of the current  d
    // (where the launchUri is), you can omit this option because the  def
    // ".". However, some servers do not support directory indexes so  "."
    // will not automatically map to the "index.html" file in that  direct
    redirectUri: "index.html"
});
```

SMART on FHIR walkthrough

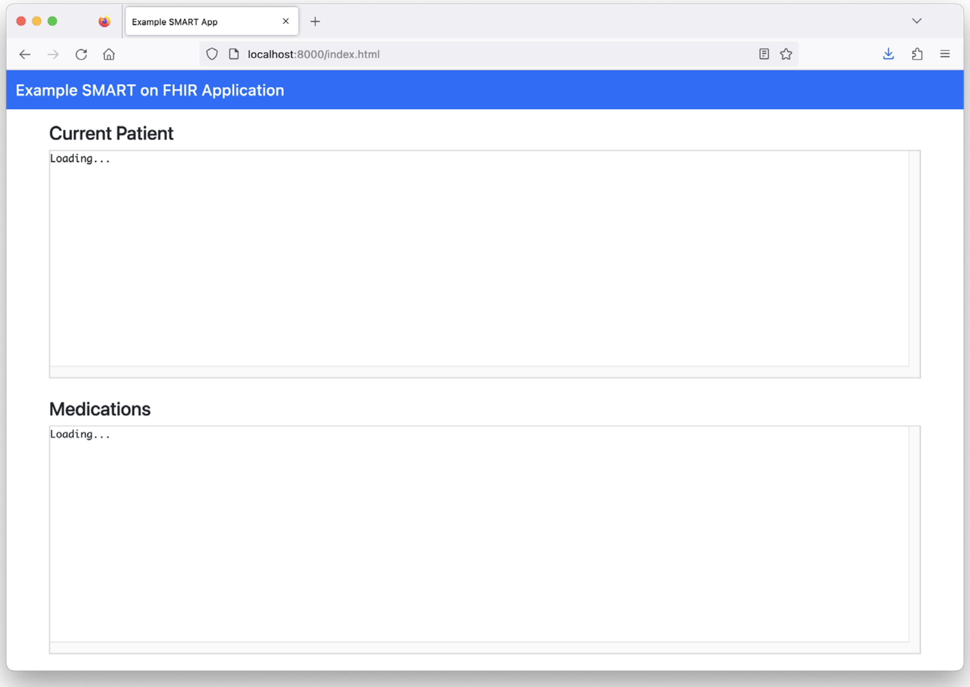
Step 3. The EHR securely authorizes (or rejects) your request.

The demo EHR had you select a provider and patient in this phase. In the real world an EHR may already pull this information from context, or show another screen specifically asking a patient to give permission for access.

SMART on FHIR walkthrough

Step 4. Your web browser gets redirected to the app's index.html

As requested earlier in your redirectUri parameter.



Example SMART on FHIR Application

Current Patient

```
Loading...
```

Medications

```
Loading...
```

SMART on FHIR walkthrough

Step 5. The app obtains an access token via FHIR.oauth2.ready()

This access token gets embedded in a client object to authenticate and authorize future FHIR queries.

```
<script type="text/javascript">
    FHIR.oauth2.ready().then(function(client)  {

        // Render the current patient (or any error)
        client.patient.read().then(
            function(pt) {
                 document.getElementById("patient").innerText = JSON.stringi
            },
            function(error) {
                 document.getElementById("patient").innerText = error.stack;
            }
        );


        client request("/MedicationRequest?patient=" + client patient id
```

SMART on FHIR walkthrough

Step 6. The app performs a Patient read FHIR query…

…and writes the raw JSON data in the app’s patient box. A real world application should parse the JSON into something more useful.

```
<script type="text/javascript">
    FHIR.oauth2.ready().then(function(client)  {

        // Render the current patient (or any error)
        client.patient.read().then(
            function(pt)  {
                document.getElementById("patient").innerText  = JSON.stringi
            },
            function(error)  {
                document.getElementById("patient").innerText  = error.stack;
            }
        );

```

SMART on FHIR walkthrough

Step 7. The app performs a MedicationRequest search query by Patient…

…and later writes the raw data in the app’s medication box.

```
        document.getElementById( patient ).innerText = JSON.string
    },
    function(error) {
        document.getElementById("patient").innerText = error.stack;
    }
);

// Get MedicationRequests for the selected patient
client.request("/MedicationRequest?patient=" + client.patient.id, {
    resolveReferences: [ "medicationReference" ], graph: true

})

// Reject if no MedicationRequests are found
.then(function(data) {
    if (!data.entry || !data.entry.length) {

```

FHIR API

FHIR API basics

- Generally speaking the pattern for a RESTful GET query appended to a URL will take the form of:

```
VERB [base]/[Resource] {?param=[value]}
```

- Spec: https://hl7.org/fhir/R4/http.html

Aside: utility of open endpoint + synthetic data

- Real-world FHIR servers will require authentication/authorization
  - Handled by the fhirclient library in our example
- But it can be helpful to have an open testing server (with synthetic data only!)
  - Synthea synthetic data Logica Sandbox

FHIR API - try it!

- Using our Logica Sandbox open endpoint:
  - https://api.logicahealth.org/FHIRResearchSynthea/open
- This is pre-loaded with Synthea data from https://synthea.mitre.org/downloads
- “Playground” JavaScript:

  https://purl.org/fhir-for-research/workshops/smart-on-fhir/playground

FHIR API - getting more data

- FHIR breaks up health information into chunks of data (resources) which are connected together via references
  - More information: https://purl.org/fhir-for-research/key- fhir-resources
  - List of all resources: https://www.hl7.org/fhir/resourcelist.html
- GET [base]/Patient/1234 retrieves an instance of the Patient resource

FHIR API - getting more data

- MedicationRequest has medication information, and is connected to Patient via MedicationRequest.subject
- GET [base]/MedicationRequest?subject=1234 will get the instances of MedicationRequest for Patient/1234
- Results are returned in an instance of Bundle

FHIR API - chaining

- MedicationRequest.subject
- has a reference back to Patient, allowing us to retrieve instances if we know the patient’s ID
- What if you only know the patient’s last name?
  - We could do two queries: one to get the ID with GET [base]/Patient? name=peter, and then a second to get the MedicationRequests for patients with that ID
  - The FHIR API supports just one query: GET [base]/MedicationRequest? subject.name=peter
  - Note that MedicationRequest.subject
  - [can be either a Patient or Group](), so this is better: GET [base]/MedicationRequest?subject:Patient.name=peter

FHIR API - reverse chaining

- What about “patients diagnosed with a given condition”?
  - The Condition resource references a Patient (or Group) in `Condition.subject`
  - The `_has` parameter supports retrieving Patients based on a value from a Condition
    - `:` separates fields

      Sub-parameters:

      - The resource type to search for references back from (`Condition`)
      - The field on that resource which would link back to the current resource (`subject`)
  - A field on that resource to filter by (`code`, which Condition uses to identify the condition) Example: `GET [base]/Patient?_has:Condition:subject:code=195662009`

FHIR API - chaining documentation

https://hl7.org/fhir/search.html#chaining

FHIR API - searching multiple values

- Logical AND to find john smith: GET [base]/Patient?given=john&family=smith
- Logical OR to find john smith or jenny smith: GET [base]/Patient? given=john,jenny&family=smith
- *Lots* more in the spec: https://hl7.org/fhir/search.html#combining

US Core

US Core

- FHIR implementation of U.S. Core Data for Interoperability (USCDI)
- Conformance to US Core is part of ONC's EHR certification program, so adoption is wide-spread in production EHRs
- Review the spec to understand available data elements: https://www.hl7.org/fhir/us/core/
  - How to read FHIR specs: https://purl.org/fhir-for- research/data-modeling-reading-igs

FHIRPath

FHIRPath

- http://hl7.org/fhirpath/:

  FHIRPath is a path based navigation and extraction language, somewhat like XPath

- Useful for extracting data from FHIR’s deeply nested data structure
- JavaScript implementation: https://github.com/HL7/fhirpath.js
  - Sandbox: https://hl7.github.io/fhirpath.js/

    Not for use with real patient data!

FHIRPath examples

Try in the sandbox: https://hl7.github.io/fhirpath.js/

- Get the value from `Patient.gender`:
  `Patient.gender`
- Get a patient's legal last name:
  `Patient.name.where(use='official').family`
- Get a patient's MRN:

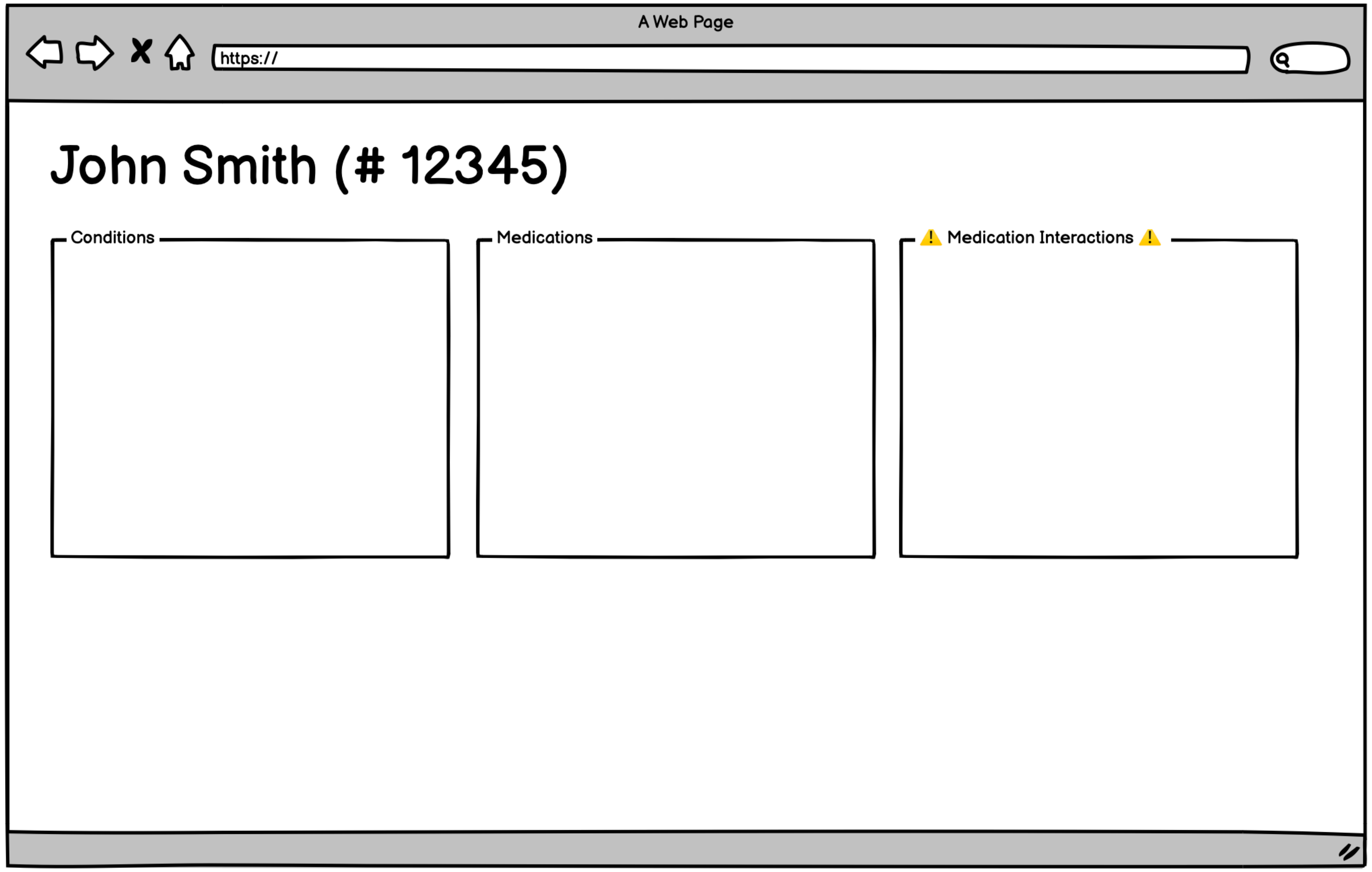
  ```
  Patient.identifier.where(type.coding.system =
  'http://hl7.org/fhir/v2/0203' and type.coding.code =
  'MR').value
  ```

Hands-on: make your own SMART on FHIR app

Hands-on prompt

- Create a decision support tool to identify medication interactions for a given patient
- Assume this will be launched via clicking a button in the EHR (SMART App Launch)
  - Could be embedded in an `<iframe>` to show inline as well – it's the same mechanism
- Use FHIR to retrieve the MedicationRequest instances for a given patient
- Use the RXNorm API to check for drug/drug interactions Display the patient's conditions, medications, and flag any drug/drug interactions

Hands-on mock-up



John Smith (# 12345)

Hands-on additional feature ideas

- Enhanced patient info (add DOB, allergies, etc.)
- Show providers who requested medication with interactions
- Group medications and drug/drug interactions by encounter

Wrap-up

Additional Resources

- Our FHIR for Research website
  docs.smarthealthit.org has:
  - Tutorials
  - Test environments
  - Vendor sandboxes
  - Sample apps
- The official SMART App Launch implementation guide
- The chat.fhir.org (Zulip) SMART stream (free account required)
  The community mailing list