Article
Lucas Enard · Sept 12, 2022 9m de lecture

Open Exchange

Un exemple simple d'un client Fhir en java

1. Fhir-client-java

Ceci est un client fhir simple en java pour s'exercer avec les ressources fhir et les requêtes CRUD vers un serveur fhir.

Notez que pour la majeure partie, l'autocomplétion est activée.

GitHub

2. Préalables

Assurez-vous que git et Docker desktop sont installé.

Déjà installé dans le conteneur :

Hapi Fhir modèle et client

3. Installation

3.1. Installation pour le développement

Clone/git tire le repo dans n'importe quel répertoire local, par exemple comme indiqué ci-dessous :

```
git clone https://github.com/LucasEnard/fhir-client-java.git
```

Ouvrez le terminal dans ce répertoire et exécutez :

```
docker build .
```

3.2. Portail de gestion et VSCode

Ce référentiel est prêt pour CodeVS.

Ouvrez le dossier fhir-client-java cloné localement dans VS Code.

Si vous y êtes invité (coin inférieur droit), installez les extensions recommandées.

3.3. Avoir le dossier ouvert à l'intérieur du conteneur

Vous pouvez être *à l'intérieur* du conteneur avant de coder si vous le souhaitez.

Pour cela, il faut que docker soit activé avant d'ouvrir VSCode.

Ensuite, dans VSCode, lorsque vous y êtes invité (coin inférieur droit), rouvrez le dossier à l'intérieur du conteneur afin de pouvoir utiliser les composants python qu'il contient.

La première fois que vous effectuez cette opération, cela peut prendre plusieurs minutes, le temps que le conteneur soit préparé.

Si vous n'avez pas cette option, vous pouvez cliquer dans le coin inférieur gauche et cliquer sur Ouvrir à nouveau dans le conteneur puis sélectionner De Dockerfile

Plus d'informations ici

En ouvrant le dossier à distance, vous permettez à VS Code et à tous les terminaux que vous ouvrez dans ce dossier d'utiliser les composants java dans le conteneur.

4. Serveur FHIR

Pour réaliser cette présentation, vous aurez besoin d'un serveur FHIR.

Vous pouvez soit utiliser le vôtre, soit vous rendre sur le site Essai FHIR gratuit d'InterSystems et suivre les étapes suivantes pour le configurer.

En utilisant notre essai gratuit, il suffit de créer un compte et de commencer un déploiement, puis dans l'onglet Overview vous aurez accès à un endpoint comme https://fhir.000000000.static-test-account.isccloud.io que nous utiliserons plus tard.

Ensuite, en allant dans l'onglet d'informations d'identification Credentials, créez une clé api et enregistrez-la quelque part.

C'est maintenant terminé, vous avez votre propre serveur fhir pouvant contenir jusqu'à 20 Go de données avec une

mémoire de 8 Go.

5. Présentation pas à pas

La présentation pas à pas du client se trouve à src/java/test/Client.java.

Le code est divisé en plusieurs parties, et nous allons couvrir chacune d'entre elles ci-dessous.

5.1. Partie 1

Dans cette partie, nous connectons notre client à notre serveur en utilisant Fhir.Rest.

```
// Partie 1

      // Créer un contexte en utilisant FHIR R4
      FhirContext ctx = FhirContext.forR4();

      // créer un en-tête contenant la clé api pour le httpClient
      Header header = new BasicHeader("x-api-key", "api-key");
      ArrayList<Header> headers = new ArrayList<Header>();
      headers.add(header);

      // créer un constructeur httpClient et lui ajouter l'en-tête
      HttpClientBuilder builder = HttpClientBuilder.create();
      builder.setDefaultHeaders(headers);

      // créer un httpClient à l'aide du constructeur
      CloseableHttpClient httpClient = builder.build();

      // Configurer le httpClient au contexte en utilisant la fabrique
      ctx.getRestfulClientFactory().setHttpClient(httpClient);

      // Créer un client
      IGenericClient client = ctx.newRestfulGenericClient("url");
```

Afin de vous connecter à votre serveur, vous devez modifier la ligne :

```
Header header = new BasicHeader("x-api-key", "api-key");
```

Et cette ligne aussi :

```
IGenericClient client = ctx.newRestfulGenericClient("url");
```

L'url' est un point de terminaison tandis que l'"api-key" est la clé d'api pour accéder à votre serveur.

Notez que si **vous n'utilisez pas** un serveur InterSystems, vous pouvez vérifier comment autoriser vos accès si nécessaire.

Comme ça, nous avons un client FHIR capable d'échanger directement avec notre serveur.

5.2. Partie 2

Dans cette partie, nous créons un Patient en utilisant Fhir.Model et nous le complétons avec un HumanName, en suivant la convention FHIR, use et family sont des chaînes et given est une liste de chaînes. De la même manière, un patient peut avoir plusieurs HumanNames, donc nous devons mettre notre HumanName dans une liste avant de le mettre dans notre patient nouvellement créé.

```
// Partie 2

      // Créer un patient et lui ajouter un nom
      Patient patient = new Patient();
      patient.addName()
         .setFamily("FamilyName")
         .addGiven("GivenName1")
         .addGiven("GivenName2");

      // Voir aussi patient.setGender ou setBirthDateElement

      // Créer le patient ressource sur le serveur
        MethodOutcome outcome = client.create()
         .resource(patient)
         .execute();

      // Enregistrez l'ID que le serveur a attribué
      IIdType id = outcome.getId();
      System.out.println("");
      System.out.println("Created patient, got ID: " + id);
      System.out.println("");
```

Après cela, nous devons sauvegarder notre nouveau Patient sur notre serveur en utilisant notre client.

Notez que si vous lancez Client.java plusieurs fois, plusieurs Patients ayant le nom que nous avons choisi seront créés.
C'est parce que, suivant la convention FHIR, vous pouvez avoir plusieurs Patients avec le même nom, seul l' id est unique sur le serveur.

Vérifier the doc pour avoir plus d'information.

Nous conseillons donc de commenter la ligne après le premier lancement.

5.3. Partie 3

Dans cette partie, nous avons un client qui recherche un patient nommé d'après celui que nous avons créé précédemment.

```
// Partie 3

      // Rechercher un patient unique avec le nom de famille exact "NomDeFamille" et
le prénom exact "Prénom1"
      patient = (Patient) client.search()
         .forResource(Patient.class)
```

```
         .where(Patient.FAMILY.matchesExactly().value("FamilyName"))
         .and(Patient.GIVEN.matchesExactly().value("GivenName1"))
         .returnBundle(Bundle.class)
         .execute()
         .getEntryFirstRep()
         .getResource();

      // Créer une télécommunication pour le patient
      patient.addTelecom()
         .setSystem(ContactPointSystem.PHONE)
         .setUse(ContactPointUse.HOME)
         .setValue("555-555-5555");

      // Changer le prénom du patient en un autre
      patient.getName().get(0).getGiven().set(0,  new StringType("AnotherGivenName"))
;

      // Mettre à jour le patient de ressource sur le serveur
      MethodOutcome outcome2 = client.update()
         .resource(patient)
         .execute();
```

Une fois que nous l'avons trouvé, nous ajoutons un numéro de téléphone à son profil et nous changeons son prénom en un autre.

Maintenant nous pouvons utiliser la fonction de mise à jour de notre client pour mettre à jour notre patient sur le serveur.

5.4. Partie 4

Dans cette section, nous voulons créer une observation pour notre patient. Pour ce faire, nous avons besoin de son identifiant, qui est son identifiant unique.

A partir de là, nous remplissons notre observation et ajoutons comme sujet, l'identifiant de notre Patient.

```
// Partie 4

      // Créer un CodeableConcept et le remplir
      CodeableConcept codeableConcept = new CodeableConcept();
      codeableConcept.addCoding()
         .setSystem("http://snomed.info/sct")
         .setCode("1234")
         .setDisplay("CodeableConceptDisplay");

      // Créer une quantité et la remplir
      Quantity quantity = new Quantity();
      quantity.setValue(1.0);
      quantity.setUnit("kg");

      // Créer une catégorie et la remplir
      CodeableConcept category = new CodeableConcept();
      category.addCoding()
         .setSystem("http://snomed.info/sct")
         .setCode("1234")
         .setDisplay("CategoryDisplay");
```

```
      // Créer une liste de CodeableConcepts et y placer des catégories
      ArrayList<CodeableConcept> codeableConcepts = new ArrayList<CodeableConcept>();
      codeableConcepts.add(category);

      // Créer une observation
      Observation observation = new Observation();
      observation.setStatus(Observation.ObservationStatus.FINAL);
      observation.setCode(codeableConcept);
      observation.setSubject(new Reference().setReference("Patient/" + ((IIdType) out
come2.getId()).getIdPart()));
      observation.setCategory(codeableConcepts);
      observation.setValue(quantity);

      System.out.println("");
      System.out.println("Created observation, reference : " + observation.getSubject
().getReference());
      System.out.println("");

       // Créer l'observation de ressource sur le serveur
      MethodOutcome outcome3 = client.create()
         .resource(observation)
         .execute();

      // Imprimer la réponse du serveur
      System.out.println("");
      System.out.println("Created observation, got ID: " + outcome3.getId());
      System.out.println("");
```

Ensuite, nous enregistrons notre observation à l'aide de la fonction de création.

5.5. Conclusion de la présentation

Si vous avez suivi ce parcours, vous savez maintenant exactement ce que fait Client.java, vous pouvez le lancer et vérifier votre Patient et votre Observation nouvellement créés sur votre serveur.

Pour le lancer, ouvrez un terminal VSCode et entrez :

```
dotnet run
```

Vous devriez voir des informations sur le Patient créé et son observation.

Si vous utilisez un serveur Intersystems, allez à API Deployement, autorisez-vous avec la clé api et d'ici vous pouvez OBTENIR par id le patient et l'observation que nous venons de créer.

6. Comment commencer le codage

Ce référentiel est prêt à être codé dans VSCode avec les plugins InterSystems.
Ouvrez Client.java pour commencer à coder ou utiliser l'autocomplétion.

7. Ce qu'il y a dans le référentiel

7.1. Dockerfile

Un dockerfile pour créer un dot net env pour que vous puissiez travailler.

Utilisez docker build . pour construire et rouvrir votre fichier dans le conteneur pour travailler à l'intérieur de celui-ci.

7.2. .vscode/settings.json

Fichier de paramètres.

7.3. .vscode/launch.json

Fichier de configuration si vous voulez déboguer

#FHIR #Java #REST API #Autre #Open Exchange #VSCode
Voir l'application sur InterSystems Open Exchange

URL de la source:https://fr.community.intersystems.com/post/un-exemple-simple-dun-client-fhir-en-java