
Article

[Lorenzo Scalese](#) · Sept 7, 2022 7m de lecture

[Open Exchange](#)

Bibliothèque de SMART sur FHIR JS et les exemples dans iris-on-fhir

Introduction

Dans ce premier article, un tutoriel simple vous a aidé à mettre en place votre déploiement FHIRaaS.

Maintenant, allons plus loin et présentons une bibliothèque JS pour accéder à la ressource FHIR.

À la fin, deux exemples d'utilisation de cette bibliothèque seront présentés, explorant le type de ressource FHIR Appointment pour les rendez-vous.

SMART sur la bibliothèque FHIR JavaScript

FHIR est une API REST, vous pouvez donc utiliser n'importe quel client HTTP pour l'utiliser. Mais c'est toujours une bonne idée d'avoir de l'aide.

En outre, il existe une initiative intéressante appelée [SMART](#) qui vise à fournir des normes pour un accès facile aux données de santé stockées dans des systèmes de santé. Cette API s'appuie sur FHIR comme couche de communication. C'est ce qu'on appelle les SMART sur les applications FHIR.

L'équipe SMART a développé une bibliothèque de client JS, qui nous aide à créer des applications compatibles avec les normes SMART. Par exemple, pour initier le flux de code d'autorisation OAuth2, vous avez juste besoin de ce code.

```
FHIR.oauth2.authorize({  
  "client_id": "<your-app-client-id>",  
  "clientSecret": "<your-app-client-secret>",  
  "scope": "openid profile user/*.*",  
  "redirect_uri": "http://localhost:64755/iris-on-fhir/index.html",  
  "iss": "https://fhirauth.lrwvcusn.static-test-account.isccloud.io/oauth2"  
});
```

Remarque : ce code est fourni pour que vous le copiez et le copiez dans l'onglet OAuth 2.0 :

portal.trial.isccloud.io/deployments/lrwwcush/oauth

JOSE PEREIRA

Accelerator Service Callback: <https://portal.trial.isccloud.io/callback>

Development Examples

FHIR.JS ANGULAR-OAUTH2-OIDC

SMART-ON-FHIR/CLIENT-JS

```
FHIR.oauth2.authorize({
  "client_id": "[REDACTED]",
  "clientSecret": "[REDACTED]",
  "scope": "openid profile user/*.*",
  "redirect_uri": "http://localhost:64755/iris-on-fhir/index.html",
  "iss": "https://fhirauth.lrwwcush.static-test-account.isccloud.io/oauth2"
});
```

POSTMAN COLLECTION GENERATE TOKEN

DELETE

© 1996-2021 – InterSystems® Corporation, Cambridge, MA Help

Afin de respecter la norme SMART, vous devez placer ce code dans une page appelée launch.html. Ainsi, un lanceur SMART sera capable de trouver le lanceur de votre application.

Notez que la méthode authorize() fait beaucoup de travail pour nous :

- Initialise le flux OAuth, en amenant l'utilisateur à la page de connexion de l'IdP.
- Si la connexion est réussie, stocke le jeton d'autorisation.
- Appelle l'URI de rafraîchissement du jeton, lorsque le jeton expire.
- Une fois qu'un jeton est récupéré, cette méthode l'utilise jusqu'à son expiration, mais vous pouvez ignorer toutes ces étapes

Une autre caractéristique intéressante de cette bibliothèque est qu'il suffit de fournir la ressource FHIR, et elle sélectionne automatiquement les chemins d'accès à l'API. Par exemple, pour obtenir des patients, il suffit d'utiliser la méthode request et de spécifier le type de ressource Patient :

```
FHIR.oauth2.ready()
  .then(client => client.request("Patient/1223"))
  .then(resource => console.log(resource))
  .catch(console.error);
```

Cette commande récupère une ressource de type Patient. Ainsi, si vous modifiez cette ressource, vous pouvez l'éditer ou même en créer une autre à partir de ce patient, par les méthodes de mise à jour et d'édition, respectivement :

```
// enregistre les changements dans la ressource patient
FHIR.oauth2.ready()
  .then(client => client.update(resource))
  .then(() => console.log('patient updated'))
  .catch(console.error);

// crée une nouvelle ressource patient
FHIR.oauth2.ready()
  .then(client => client.create(resource))
```

```
.then(() => console.log('patient created'))  
.catch(console.error);
```

Notez que vous n'avez pas spécifié de chemin d'accès à l'API FHIR, car la bibliothèque résout le chemin d'accès correct à l'API en fonction du type de la ressource.

Enfin, pour supprimer une ressource :

```
FHIR.oauth2.ready()  
  .then(client => client.delete(`Patient/1223`))  
  .then(() => console.log('patient deleted'))  
  .catch(console.error);
```

Ressource FHIR pour les rendez-vous

Afin de montrer l'utilisation d'une autre ressource, créons une simple application de rendez-vous en manipulant la ressource FHIR Appointment.

Veuillez noter qu'il s'agit d'un cas d'utilisation très simple de cette ressource, qui n'utilise que quelques-unes de ses propriétés. En outre, cette ressource est liée à d'autres, qui ne sont pas couvertes ici. Si vous souhaitez en savoir plus sur ces ressources et le flux de travail entre elles, vous devez vous reporter à la [spécification HL7 FHIR](#).

Tout d'abord, un objet modèle est récupéré à partir de la ressource API Development for Appointment. Cet objet est utilisé pour fournir un prototype pour la création et l'édition de rendez-vous.

```
function getAppointmentTemplate() {  
  return {  
    "resourceType": "Appointment",  
    "id": "examplereq",  
    ...  
  };  
}
```

Ensuite, une fonction permettant de modifier le prototype est créée. Cette fonction reçoit le prototype de ressource Appointment de rendez-vous et une instance d'un événement de calendrier. Cet objet d'événement contient des propriétés remplies par l'utilisateur via un composant de calendrier, qui seront introduites dans le prototype de ressource.

```
function setEventDataToResource(resource, eventData) {  
  if (eventData.id) {  
    resource.id = eventData.id;  
  }  
  
  const patient = getSelectedPatientObject();  
  resource.participant[0].actor.reference = `Patient/${patient.id}`;  
  resource.participant[0].actor.display = patient.name[0].given[0];  
  resource.description = eventData.title;  
  resource.comment = eventData.extendedProps.description;  
  if (!resource.comment) {  
    delete resource.comment;  
  }  
}
```

```
if (eventData.allDay) {
  eventData.start = eventData.start.split(" ")[0];
  eventData.end = eventData.end.split(" ")[0];
}

const hasTime = {
  start: eventData.start.indexOf(" ") > -1,
  end: eventData.end.indexOf(" ") > -1
}
if (hasTime.start || hasTime.end) {
  resource.requestedPeriod[0].start = new Date(eventData.start).toISOString();
  resource.requestedPeriod[0].end = new Date(eventData.end).toISOString();
} else {
  resource.requestedPeriod[0].start = eventData.start;
  resource.requestedPeriod[0].end = eventData.end;
}

return resource;
}
```

Grâce à ces deux fonctions, nous pouvons créer et remplir correctement une ressource Rendez-vous. Maintenant, nous devons utiliser la bibliothèque SMART pour effectuer les opérations CRUD sur cette ressource, finalisant ainsi notre application simple de rendez-vous :

```
function getAppointments(client) {
  return client.request(`Appointment`);
}

function createAppointment(eventData) {
  const resource = setEventDataToResource(
    getAppointmentTemplate(), eventData
  );
  return getFHIRClient()
    .then(client => {
      return client.create(resource);
    });
}

function updateAppointment(eventData) {
  const resource = setEventDataToResource(
    getAppointmentTemplate(), eventData
  );
  return getFHIRClient()
    .then(client => {
      return client.update(resource);
    });
}

function deleteAppointment(id) {
  return getFHIRClient()
    .then(client => {
      return client.delete(`Appointment/${id}`);
    });
}
```

Voici un screencast de l'application en cours d'exécution :

N'oubliez pas que le code exposé ici est simplement l'essentiel pour comprendre l'application. Mais, si vous souhaitez plus de détails, veuillez consulter le code source.

Bot Telegram

Comme dernière démonstration de ce que vous pouvez faire de cool avec FHIRaaS, je voudrais montrer un bot Telegram très simple.

J'ai entièrement basé cette fonctionnalité sur les travaux de [Nikolay Soloviev](#) et [Sergey Mikhailenko](#). Dans l'article de Nikolay, vous pouvez voir comment créer le robot dans Telegram. Consultez également l'application de Sergey Mikhailenko pour obtenir plus d'informations sur les bots Telegram.

Comme le plus gros du travail a déjà été fait par Nikolay et Serguey, j'ai juste ajouté une couche supplémentaire de code. Cette couche récupère l'identifiant du patient, ainsi que la date et l'heure du rendez-vous, auprès des utilisateurs de Telegram, dans le cadre d'une simple interaction de type chat, sans fonctionnalités NLP. Après avoir récupéré ces informations, un point de terminaison FHIRaaS est appelé pour afficher un nouveau rendez-vous.

Voici un screencast du bot en action :

Si vous souhaitez obtenir plus de détails sur la mise en œuvre, veuillez consulter le code de l'application.

Conclusion

Dans cet article, nous avons fait une présentation de la bibliothèque SMART on FHIR JavaScript Library et montré comment l'utiliser sur deux exemples basés sur le type de ressource FHIR Appointment.

Dans le prochain article, nous allons approfondir les autres fonctionnalités de l'application iris-on-fhir.

[#FHIR #OAuth2 #InterSystems IRIS for Health #Open Exchange](#)
[Voir l'application sur InterSystems Open Exchange](#)

URL de la source: <https://fr.community.intersystems.com/post/biblioth%C3%A8que-de-smart-sur-fhir-js-et-les-exemples-dans-iris-fhir>