GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT SUR SYSTÈMES OPEN SOURCE

N°249

JUIN 2021

FRANCE MÉTRO. : 8,90 €
BELUX : 9.90 € - CH : 15.10 CHF
ESP/IT/PORT-CONT : 9,90 €
DOM/S : 9,90 €
TUN : 25 TND - MAR : 100 MAD
CAN : 16,50 $CAD

L 19275 - 249 - F: 8,90 € - RD

VUI / Go

SIRI, ALEXA, GOOGLE HOME...

DÉVELOPPEZ VOTRE INTERFACE VOCALE

p.20

Code / Syntaxe

DÉCOUVREZ QUELQUES STRUCTURES PEU CONNUES DE PYTHON p.48

GDScript / UI

CRÉEZ UNE INTERFACE TACTILE POUR GODOT p.70

Outils / LaTeX

AJOUTEZ DES NOTES, DES RÉFÉRENCES ET UN GLOSSAIRE À VOS DOCUMENTS p.52

Dev / Projets

COMPRENEZ L'INTÉGRATION CONTINUE AVEC DRONECI p.36

Actualités / Multiplateforme

FLUTTER 2 : DÉCOUVREZ LA MISE À JOUR MAJEURE DU FRAMEWORK DE DEV MOBILE p.06

[MOOC] Wikipédia

Apprenez à contribuer à l'encyclopédie Wikipédia,
et découvrez les coulisses du septième site Internet le plus visité au monde !

Inscription sur formations.wikimedia.fr

Le MOOC Wikipédia est un cours en ligne gratuit et ouvert à tous·tes proposé par Wikimédia France, l'association française de promotion de la connaissance libre. Plus d'informations sur www.wikimedia.fr.

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

10, Place de la Cathédrale - 68000 Colmar - France
Tél. : 03 67 10 00 20 – **Fax :** 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com
www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Résponsable service infographie : Kathrin Scali
Réalisation graphique : Thomas Pichon
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 8,90 €

Membre April www.april.org

https://www.gnulinuxmag.com

RETROUVEZ-NOUS SUR :

@gnulinuxmagazine

@gnulinuxmag

p.41 DÉCOUVREZ TOUS NOS ABONNEMENTS PAPIER !

Codes sources sur :
https://github.com/glmf

ÉDITO

Il y avait bien longtemps que je n'avais plus pensé au laissez-passer A38… celui que l'on obtient au guichet 1, couloir de gauche, dernière porte à droite. Pourtant, cette caricature de la lourdeur administrative (« Les 12 travaux d'Astérix ») que l'on pouvait croire réservée à la paperasse et datant donc d'une autre époque est bel et bien toujours d'actualité, même avec le numérique.

Un ami m'a montré un courrier reçu d'une administration - inutile de la nommer, il y a de fortes chances pour que le cas soit reproductible - lui demandant de régulariser sa situation en complétant une déclaration. Joies de la modernisation, il n'est plus besoin de se déplacer pour quérir le sésame, les déclarations sont disponibles en ligne au format électronique ! Ceci est clairement indiqué dans le courrier :

« Les déclarations sont disponibles sur notre site via le moteur de recherche. »

Formidable ! Donc plutôt que d'indiquer qu'il faut remplir le laissez-passer A38, de donner un lien direct vers le fichier PDF ou, allez soyons fous, de donner un QR-code permettant d'accéder directement au document, on préfère laisser l'utilisateur errer sur un site contenant des centaines de formulaires pour lesquels une recherche ciblée réduira le nombre à trois ou quatre ? Mais la « maison des fous » existe et elle a changé de dimension afin de perdurer dans un monde numérique ! Une fois que l'on est en présence des trois formulaires, quel est le critère de sélection ? Peut-on s'attendre à des explications claires, compréhensibles par tout un chacun afin d'effectuer un choix ? Que nenni, le tirage aléatoire sera sans aucun doute le plus efficace !

Tout serait tellement plus simple si tout était correctement pensé en amont en vue de simplifier les démarches administratives. Nous avons les outils pour cela : il suffit de les utiliser, à condition de le vouloir ou de pouvoir...

Comme toujours GNU/Linux Magazine vous propose des articles pointus dans différents domaines. Ce mois-ci, vous pourrez découvrir les nouveautés de **Flutter 2** pour le développement de Progressive Web Apps, vous plongerez dans le monde des Voice User Interfaces, et pourrez mettre en pratique différents outils et langages avec **LaTeX**, **Python**, **DroneCI**, ou encore **Godot**, ce qui vous permettra d'éviter à vos utilisateurs de rentrer dans une nouvelle « maison des fous »...

Tristan Colombo

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE
N°249

ACTUS & HUMEUR

OUTILS & SYSTÈME

LIBS & MODULES

Chez votre marchand de journaux

et sur **www.ed-diamond.com**

PAPIER
en kiosque

FLIPBOOK HTML5
sur www.ed-diamond.com

CONNECT
LA DOCUMENTATION TECHNIQUE DES PROS DE L'IT
sur connect.ed-diamond.com

FLUTTER 2 : L'ALTERNATIVE PROFESSIONNELLE

ALAIN BASTY
[Ingénieur de développement logiciel]

MOTS-CLÉS : FLUTTER, DÉVELOPPEMENT, APPLICATION, DART, MOBILE, WEB, DESKTOP, UI

Le 3 mars 2021, Google annonce Flutter 2 dans le premier événement « Flutter Engage ». La target Web/PWA devient stable et les targets desktop deviennent bêta avec une preview sur stable. Du côté Dart, la null safety devient la norme. Pendant ce temps, « Courses » se structure, adopte MVVM et devient non nullable !

Nous avons vu dans un premier article comment installer **Flutter** et comment générer une application **Web** et **Linux [1]**. L'arrivée de Flutter 2 permet de considérer ce framework en production professionnelle, puisque la target Web/**PWA** devient stable, les targets desktop deviennent bêta et cerise qui en ferait oublier le gâteau : Flutter passe en Null Safety **[2]**, ce qui améliore la stabilité et la performance des applications, tout en rendant le code plus clair et plus concis.

Dans une première partie, nous explorerons l'application exemple qui, si elle reste très simple, a été complètement refondue pour démontrer des techniques plus modernes : utilisation des routes pour la navigation, abstraction de l'accès au stockage persistant, gestion de l'état plus performante à base de notifications.

Nous verrons ensuite les outils de contrôle et d'aide à la qualité de code intégrés à Flutter : Le passage en null safety, l'analyse statique, les tests unitaires, la génération de la documentation, tout cela en ligne de commande, prêts pour un pipeline **CI/CD [3]** !

1. DE MVC À MVVM

1.1 Les temps modernes

Dans l'article précédent **[1]**, nous avons prototypé une application de liste de courses en Flutter qui gère et manipule des données locales selon une architecture **MVC**. Aujourd'hui, nos terminaux sont tous connectés et une application n'est plus qu'un simple frontend à des logiciels backend distants, responsables de la gestion de la concurrence entre de multiples utilisateurs, des politiques d'accès aux données et aux fonctionnalités, et qui implémentent des logiques métiers complexes et distribuées sur le réseau.

Nous allons donc préparer notre application au monde moderne en utilisant le pattern d'architecture « Modèle - Vue - Vue Modèle » (**MVVM**) **[4]** qui sépare principalement la logique de présentation, le front office, de la logique métier et ses données ou back office. Dans cette architecture, c'est le « Vue Modèle » local qui gère l'état de l'application, notifie des changements d'état à la « Vue », et qui se synchronise avec le « Modèle » indépendant de l'application.

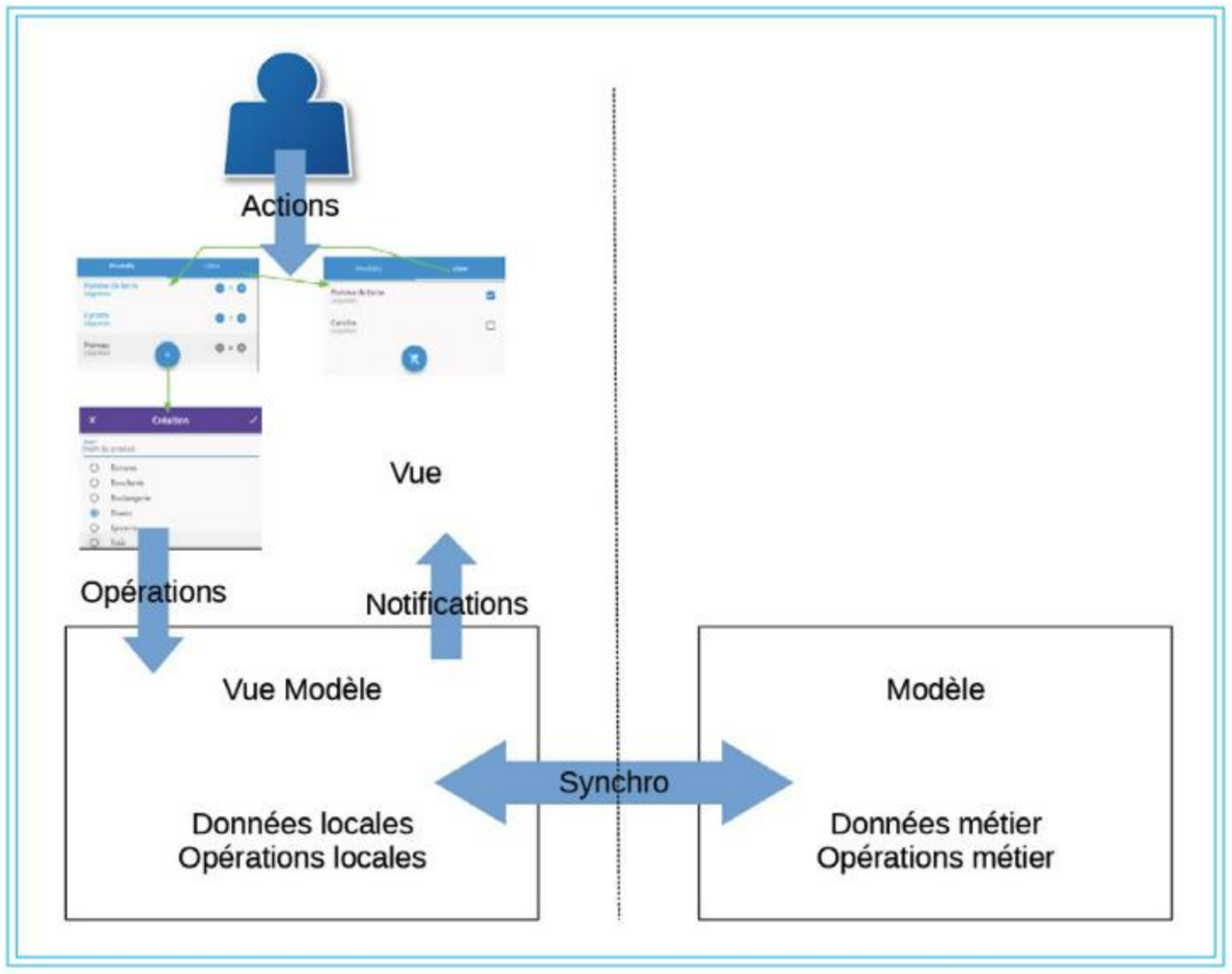


Fig. 1 : MVVM : à gauche le Frontend, à droite le Backend.

Dans cet exemple, nous n'avons pas de véritable logique métier distante, mais seulement l'accès à un stockage physique local. Cependant, conformément à MVVM, notre « Vue Modèle » est devenu indépendant de la logique de stockage et nous générons artificiellement une latence en lecture afin de simuler un accès lent à un hypothétique réseau (figure 1). L'application propose en outre une UI de substitution à son lancement, tant que les données ne sont pas chargées.

Le code a donc subi de nombreux changements. Il a été éclaté en différents composants, on a ajouté des tests automatiques, de la documentation, on est passé en null safety (tableau ci-dessous).

Fichier	Description	Interfaces
`lib/main.dart`	Le point d'entrée et les initialisations globales.	`main()`
`lib/modele.dart`	Le « Vue Modèle » et les classes associées.	`VueModele`, `Produit`, `Rayon`
`lib/storage.dart`	Les classes d'accès au stockage des données.	`StorageStrategy`, `MemoryMapStrategy`, `LocalStorageStrategy`, `DelayedStrategy`
`lib/liste_screen.dart`	L'écran présentant la liste des produits.	`ListeScreen`, `ProduitConsumer`
`lib/produit_screen.dart`	L'écran d'édition / création d'un produit.	`ProduitScreen`
`test/modele_test.dart`	Les tests unitaires du modèle et du contrôleur.	
`test/widget_test.dart`	Les tests de widgets.	
`test/dataset.dart`	Les données pour les tests.	`dataset1`

Pour exécuter cette nouvelle version, on revient sur le channel stable de Flutter et on clone le source de **courses2** :

```
$ flutter channel stable
$ flutter upgrade
$ git clone git@github.com:abasty/courses2.git
$ cd courses2
$ rm -rf linux
$ flutter clean
$ flutter create .
$ flutter run -d linux
💪 Running with sound null safety 💪
```

L'application a pris du muscle !

1.2 Chacun sa route

Même si notre application reste simple avec ses deux écrans **ListeScreen** et **ProduitScreen**, on utilise l'objet **Navigator**, défini par Flutter au niveau de **MaterialApp**, pour centraliser la définition des routes menant vers ces écrans. On peut s'imaginer les routes comme des URL vers les différentes pages d'une application.

Les routes sont nommées de manière globale et on a choisi de définir ces noms avec des constantes statiques dans chaque classe screen :

```
class ListeScreen extends StatelessWidget {
  static const name = '/liste';
...
class ProduitScreen extends StatefulWidget {
  static const name = '/produit';
  final Produit? _init;

  ProduitScreen(this._init);
...
```

On remarque au passage que le constructeur **ProduitScreen** prend un **Produit**, possiblement **null**, en argument. S'il est **null**, c'est qu'on veut créer un nouveau produit, sinon c'est le produit qu'on souhaite éditer.

On configure les routes lorsqu'on crée la **MaterialApp** dans **main()** :

```
MaterialApp(
  routes: {
    ListeScreen.name: (context) => ListeScreen(),
  },
  onGenerateRoute: (RouteSettings settings) {
    if (settings.name == ProduitScreen.
name) {
      return MaterialPageRoute(
        builder: (context) =>
ProduitScreen(settings.arguments as
Produit?),
      );
    }
    return null;
  },
  initialRoute: ListeScreen.name,
```

routes est une **Map**, un tableau associatif, l'équivalent d'un dictionary **Python**, qui associe chaque nom de route à un **WidgetBuilder**, une callback qui construit un widget. Dans l'exemple, le nom de la route **/liste** engendre la création implicite d'un objet **MaterialPageRoute** avec comme builder la fonction anonyme **(context) => ListeScreen()** qui construit le widget **ListeScreen**.

Pour avoir une plus grande maîtrise sur la création d'une **MaterialPageRoute** on peut, au lieu de déclarer la route dans le tableau **routes**, exécuter du code sur **onGenerateRoute**. Dans l'exemple, quand le nom de la route est **/produit**, on crée explicitement une **MaterialPageRoute** dont le builder a accès aux settings et donc aux arguments de la route. On peut ainsi passer un produit en paramètre au constructeur du widget **ProduitScreen**.

Finalement, avec **initialRoute**, on définit la route qui est invoquée automatiquement lors du lancement de l'application, ici **/liste** qui conduit à la page **ListeScreen**.

Les routes permettent ainsi de naviguer dans les différents écrans d'une application en les invoquant simplement dans le code. Lors d'un long press sur un produit, on empile la route **/produit** avec le produit à éditer en argument (figure 2) :

```
Widget _produitsTabTile(BuildContext
context, Produit p, Widget? child) {
  ...
  onLongPress: () => Navigator.pushNamed(
    context,
    ProduitScreen.name,
    arguments: p,
```

Lors de l'ajout d'un nouveau produit, avec l'action button ***(+)***, on ne renseigne pas **arguments**, qui restera à sa valeur par défaut de **null** :

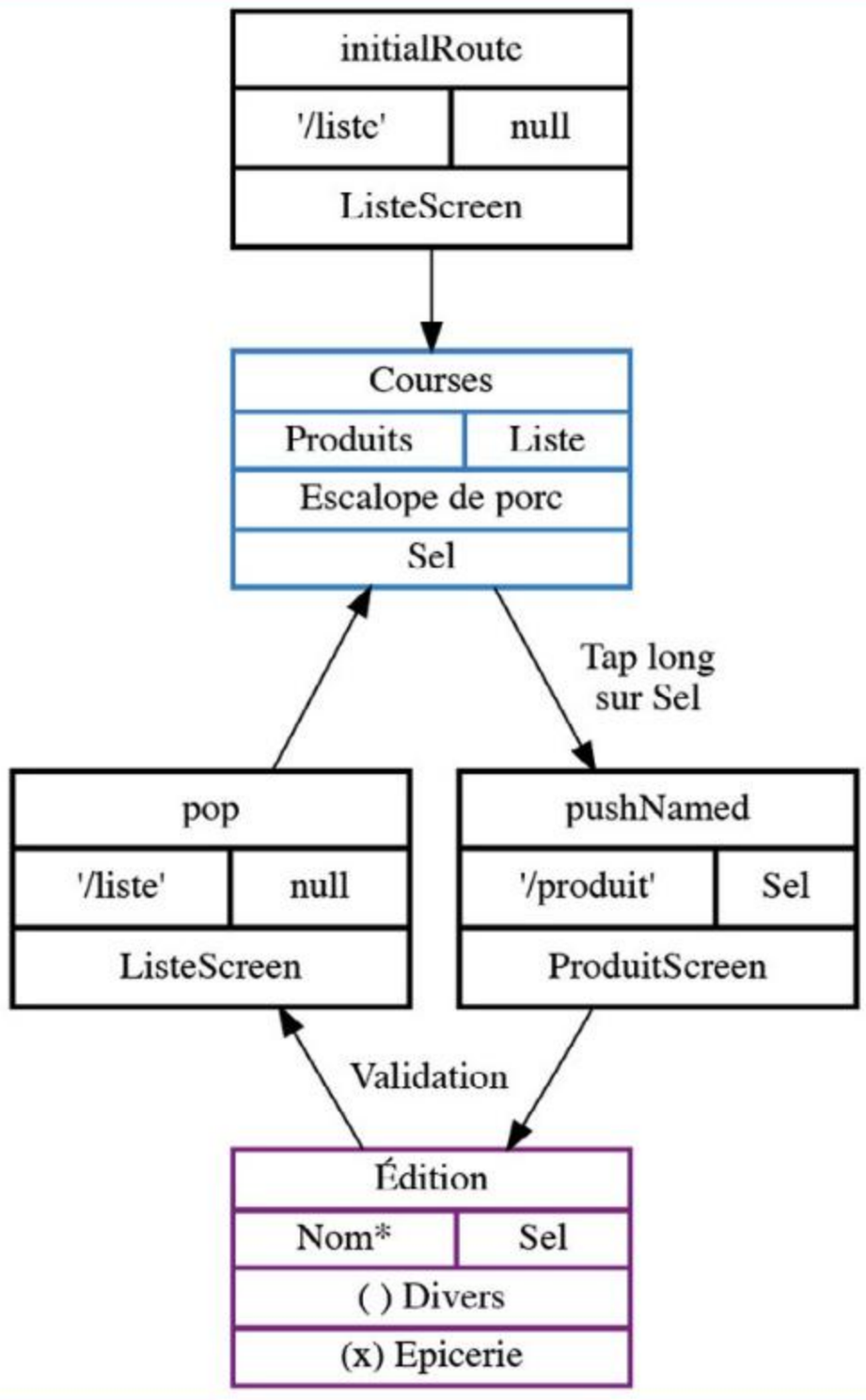


Fig. 2 : Exemple d'édition du produit "Sel".

```
Widget _produitsTab() {
...
  LocalActionButton(
    Icons.add,
    () => Navigator.pushNamed(
      context,
      ProduitScreen.name,
```

Enfin, lorsqu'on annule ou valide l'écran d'édition d'un produit, on dépile la route courante et on revient donc sur **`/liste`** :

```
leading: IconButton(
  icon: Icon(Icons.clear),
  onPressed: () => Navigator.pop(context),
```

1.3 Storage, ô des espoirs !

Les données de l'application sont conservées en mémoire dans l'objet **`modele`** qui référence des objets **`Produit`** et **`Rayon`**, comme le montre la figure 3. Au lancement de l'application, ces données sont lues avec **`loadAll()`** depuis un storage physique. Chaque fois qu'on modifie le modèle à l'aide des méthodes de contrôle **`ctrl*`**, les données sont sauvegardées dans le storage avec **`saveAll()`**. Afin d'abstraire l'accès au support physique, on a détaché le modèle de la logique de stockage. La classe abstraite **`StorageStrategy`** définit une interface d'accès à un storage suivant le design pattern « Strategy » **[5]** :

```
abstract class StorageStrategy {
  Future<void> write(Map<String, dynamic> json);
  Future<Map<String, dynamic>> read();
}
```

`LocalStorageStrategy` et **`MemoryMapStrategy`** sont deux classes qui implémentent **`StorageStrategy`**. Lors de la création du modèle, on passe simplement en paramètre la stratégie de stockage à utiliser. Dans l'application, on a **`modele = VueModele(LocalStorageStrategy())`** qui fournit un accès au système de fichiers pour les targets desktop et mobile ou au local storage pour la target Web. Dans les tests, on utilise **`modele = VueModele(MemoryMapStrategy(dataset1))`** qui fournit un accès à une **`Map`** en mémoire. Lorsqu'on implémentera l'accès à un backend, il « suffira » d'implémenter une nouvelle stratégie d'accès aux données.

Les données échangées entre le modèle et le storage sont structurées à l'aide de **`Map<String, dynamic>`**, c'est-à-dire de tableaux associant des clés textuelles à des valeurs d'un type de base comme **`String`**, **`int`** ou **`bool`**, d'un type **`Map<String, dynamic>`** ou d'une liste **`List<Map<String, dynamic>>`**. Les objets **`Map`** sont très faciles à manipuler en **Dart** et sont syntaxiquement proches de **JSON** :

```
const dataset1 = {
  'rayons': [
    {'nom': 'Divers'},
    {'nom': 'Boucherie'},
    {'nom': 'Légumes'},
    ...
  ],
  'produits': [
    {
      'nom': 'Escalope de porc',
      'rayon': {'nom': 'Boucherie'},
     'quantite': 0,
      'fait': false
    },
    ...
  ]
};
```

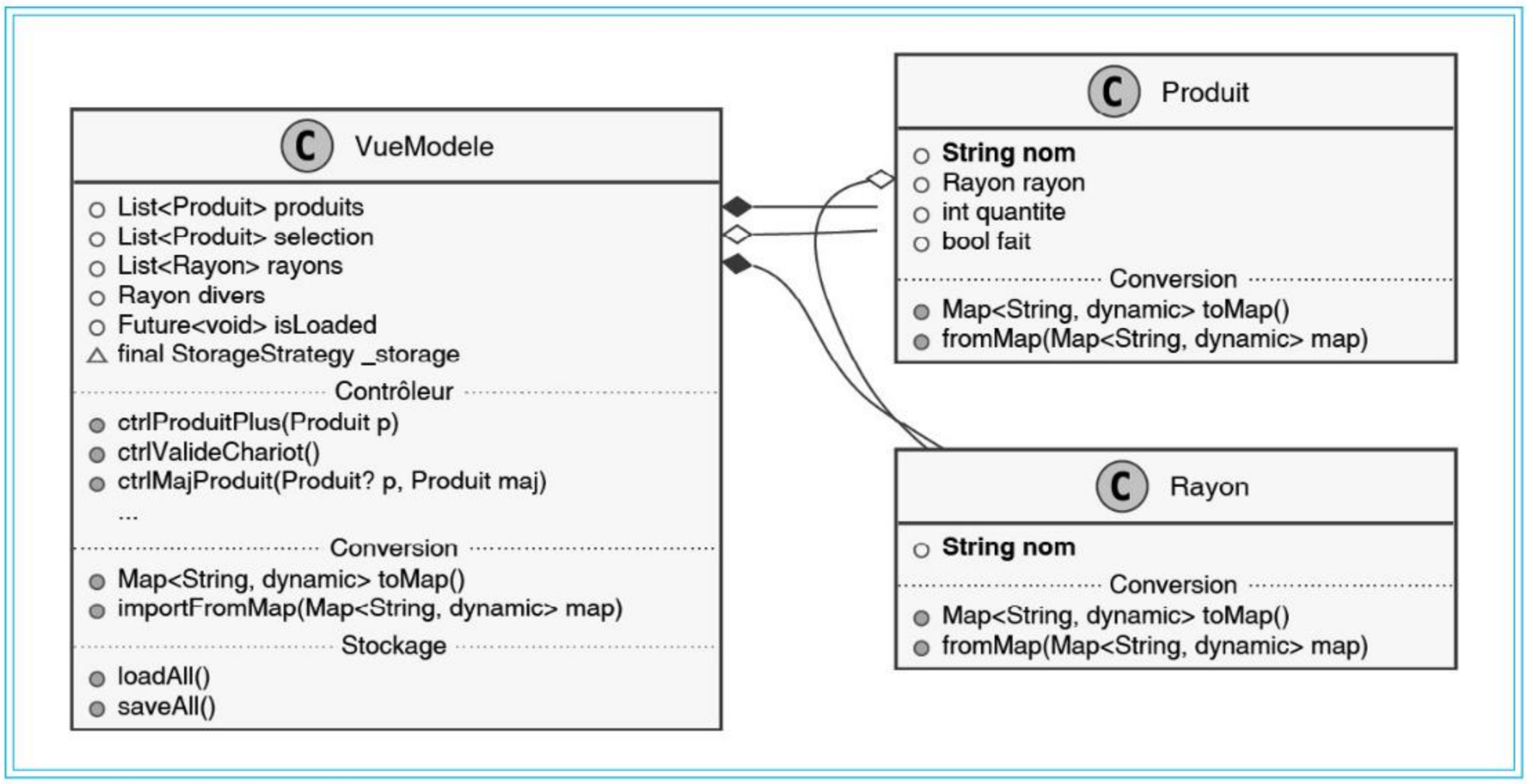


Fig. 3 : Structure de données interne.

Les méthodes **`toMap()`** des classes **`VueModele`**, **`Rayon`** et **`Produit`**, appelées lors d'un **`saveAll()`**, construisent une **`Map`** depuis les propriétés de leur classe respective :

```
/// Transforme ce [Produit] en `Map<String,
dynamic>`
Map<String, dynamic> toMap() =>
    {'nom': nom, 'rayon': rayon.toMap(),
'quantite': quantite, 'fait': fait};
```

Les constructeurs **`fromMap()`** de **`Produit`** et **`Rayon`** créent une nouvelle instance en réalisant la transformation inverse. La méthode **`importFromMap()`** de **`VueModele`** appelle ces constructeurs pour importer de nouveaux produits et rayons dans le modèle. On remarquera dans le code suivant l'utilisation du mot clé **`as`** qui permet de caster une valeur **`dynamic`** dans un type particulier. Si à l'exécution, la valeur ne peut pas être castée, une exception **`TypeError`** est levée, que l'on devrait gérer dans un bloc **`try/on`**.

```
/// Crée un nouveau [Produit] depuis une [map]
factory Produit.fromMap(Map<String, dynamic>
map) => Produit(
    map['nom'] as String,
    Rayon.fromMap(map['rayon'] as Map<String,
dynamic>),
    map['quantite'] as int,
    map['fait'] as bool);
```

Les **`Map<String, dynamic>`** rendent très simple l'implémentation de **`MemoryMapStrategy`** :

```
class MemoryMapStrategy implements
StorageStrategy {
  Map<String, dynamic> _map;

  MemoryMapStrategy(this._map);

  @override
  Future<void> write(Map<String,
dynamic> map) async => _map = map;

  @override
  Future<Map<String, dynamic>> read()
async => _map;
}
```

Quant à **`LocalStorageStrategy`**, il convertit les **`Map`** vers et depuis JSON grâce aux utilitaires de la bibliothèque intégrée **`dart:convert`** :

```
await _storage.setItem('modele', json.
encode(map));
...
var str = await _storage.
getItem('modele') as String;
return json.decode(str) as Map<String,
dynamic>;
```

1.4 Retour sur le « Future »

Il est probable que lors de l'initialisation du modèle, au lancement de l'application, le premier `loadAll()` prenne « un certain temps », surtout si on envisage de faire des requêtes réseau, comme l'appel d'API REST **[6]**. Tant que les données ne sont pas chargées, `courses2` affiche un indicateur de progression.

Pour simuler ce délai, une classe `DelayedStrategy` introduit artificiellement une temporisation après le `read` d'un véritable storage :

```
class DelayedStrategy implements
StorageStrategy {
  ...
  DelayedStrategy(this._storage, this.
_seconds);
  ...
  Future<Map<String, dynamic>> read()
async {
    var map = _storage.read();
    await Future.
delayed(Duration(seconds: _seconds));
    return map;
```

On crée le modèle avec `modele = VueModele(DelayedStrategy(LocalStorageStrategy(), 2))`. Le modèle définit le « Future » `isLoaded` qui se réalise de façon asynchrone lorsque le modèle est effectivement chargé :

```
class VueModele {
  final StorageStrategy _storage;
  late Future<void> isLoaded;

  VueModele(this._storage) {
    isLoaded = loadAll();
  }

  Future<void> loadAll() async =>
importFromMap(await _storage.read());
```

Finalement, côté UI, un `FutureBuilder` alterne entre l'indicateur de progression, `CircularProgressIndicator` et l'affichage du modèle, `_scaffold()`, lorsque `isLoaded` se réalise, c'est-à-dire quand `connectionState` devient `done` :

```
Widget build(BuildContext context) {
    return FutureBuilder(
      future: modele.isLoaded,
      builder: (context, snapshot) =>
          snapshot.connectionState ==
ConnectionState.done
```

```
                ? _scaffold()
                : Container(
                    color: Colors.white,
                    child: Center(
                      child:
CircularProgressIndicator(),
```

1.5 Une affaire d'état

Flutter propose en standard plusieurs approches de la gestion d'état d'une application. Nous avons vu l'approche `setState()`, très simple à comprendre parce que très bas niveau. Elle introduit cependant une certaine lourdeur dans le code, et l'optimisation de la construction d'un arbre de widgets nécessite la déclaration de multiple couples de classes `StatefulWidget` / `State`, les appels à `setState()` devenant vite un cauchemar. Dans `courses2`, l'écran `ProduitScreen`, très simple, conserve cette approche. `ListeScreen` s'appuie par contre sur l'approche « Provider » intégrée à Flutter. D'autres solutions sont possibles et pour les inconditionnels de **BLoC**, **Redux**, **MobX**, des packages externes sont disponibles **[7]**.

L'approche « Provider » s'appuie sur la classe `ChangeNotifier`, et deux widgets Flutter `ChangeNotifierProvider` et `Consumer` **[8]**. Le couple `ChangeNotifier` / `ChangeNotifierProvider` s'articule autour du design pattern « Observable / Observer » **[9]**, renommé en Dart Listenable / Listener. Dans la descendance d'un `ChangeNotifierProvider`, les widgets `Consumer` sont reconstruits quand leur `ChangeNotifierProvider` est notifié par le `ChangeNotifier` (figure 4).

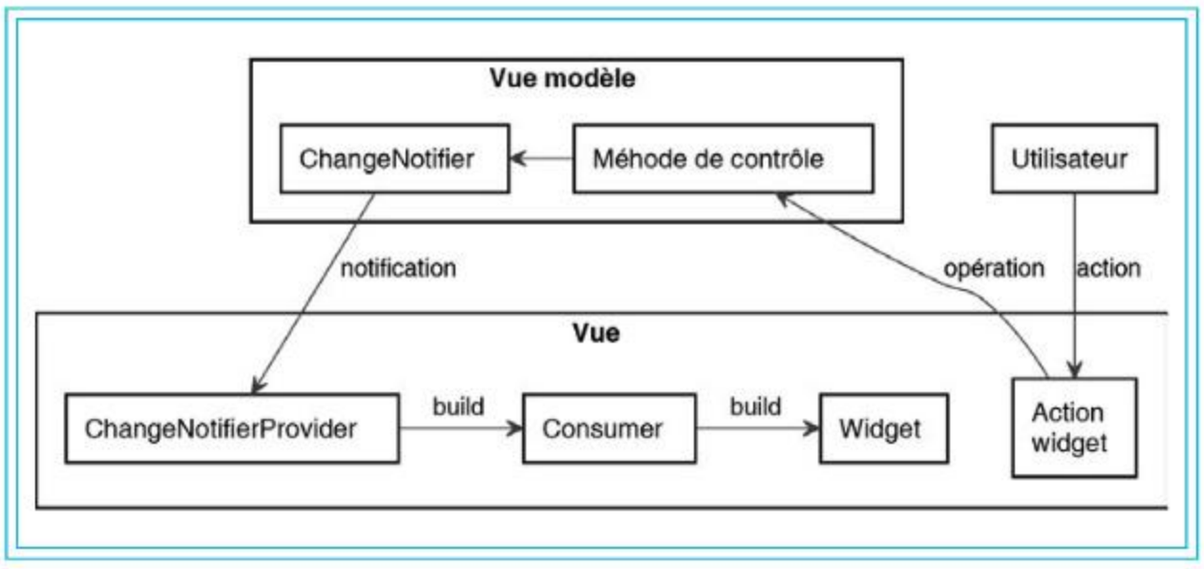


Fig. 4 : L'approche « Provider » dans une architecture MVVM.

Dans `ListeScreen`, quand l'utilisateur modifie la quantité d'un produit, seul le widget qui affiche ce produit est rebuildé. Cela optimise donc l'affichage de la page, là où précédemment

on redessinait l'application complète. **ListeScreen**, comme l'application, sont des **StatelessWidget** et aucun appel à **setState()** n'est possible.

Pour ce faire, on a rendu la classe **Produit** listenable :

```
class Produit extends ChangeNotifier
```

Quand on modifie un produit, par exemple quand on incrémente sa quantité, on notifie les listeners de ce produit :

```
void ctrlProduitPlus(Produit p) {
  p.quantite++;
  p.notifyListeners();
```

Dans la vue, chaque élément qui présente un produit doit être notifié et être rebuildé quand ce produit change. On a factorisé un **ChangeNotifierProvider** et un **Consumer** de **Produit** dans la classe **ProduitConsumer** :

```
class ProduitConsumer extends StatelessWidget {
  ...
  const ProduitConsumer(this._produit,
this._builder);

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider<Produit>.
value(
      value: _produit,
      child: Consumer<Produit>(builder:
_builder),
```

Finalement, on a placé **ProduitConsumer** dans l'onglet ***Produits*** :

```
Widget _produitsTab() {
  return ListView.builder(
    itemCount: modele.produits.length,
    itemBuilder: (context, index) =>
        ProduitConsumer(modele.produits[index],
_produitsTabTile),
```

Et aussi dans l'onglet ***Liste*** :

```
Widget _listeTab() {
  return ListView.builder(
    itemCount: modele.selection.length,
    itemBuilder: (context, index) =>
        ProduitConsumer(modele.
selection[index], _listeTabTile),
```

Quand le « Vue Modèle » est modifié plus globalement, par exemple lors d'un **loadAll()** ou de l'ajout d'un produit, un **ChangeNotifierProvider<VueModele>** placé au-dessus des onglets et deux **Consumer<VueModele>**, placés au-dessus de chacune des **ListView**, permettent de builder entièrement la liste de l'onglet courant. Évidemment, notre **VueModele** est lui aussi devenu un **ChangerNotifier**.

Quand on exécute **courses2** en debug, des messages dans la console témoignent des éléments effectivement buildés.

LE PACKAGE PROVIDER

Si ChangeNotifier fait partie des Flutter foundations, ChangeNotifierProvider et Consumer sont disponibles dans le package provider, qu'il faut donc ajouter dans le pubspec.yaml de l'application.

```
dependencies:
  flutter:
    sdk: flutter
  provider: ^5.0.0
```

Dans le source de la vue, on rajoute provider :

```
import 'package:provider/provider.dart';
```

Et dans le source du « Vue Modèle » :

```
import 'package:flutter/foundation.dart';
```

2. QUALITÉ DE CODE

2.1 Objectif non null

Dart 2.0 est arrivé en août 2018 avec un sound type system impliquant au code un typage fort des variables. En mars 2021, la release 2.12 devient stable ainsi que Flutter 2 : la sound null safety **[10]** fait son apparition. Le code doit maintenant spécifier explicitement si une variable peut recevoir la valeur **null**. Le code des packages standard Dart et Flutter est null safe et les développeurs sont incités à en faire de même. Il est toujours possible de ne pas mettre à niveau son code, mais c'est se priver d'un formidable outil d'optimisation et de prévention de bugs. Le 14 mars 2021, **courses2** devient null safe (figure 5).

Fig. 5 : Dans la documentation, un logo précise si le code est « null safe ».

L'analyseur statique de code Dart permet de vérifier et de certifier que le critère de non nullabilité est propagé dans la totalité des branches du code, dans les méthodes de toutes les bibliothèques et packages utilisés. L'analyseur ne fait aucune supposition et si une branche rompt la chaîne de non nullabilité, le code ne sera simplement pas compilable. Il est impossible de se retrouver à l'exécution avec une variable à **null** si elle est déclarée non nullable. De même, si une variable est déclarée nullable, un test par rapport à **null** sera obligatoire pour que le code puisse être compilé. C'est tout simplement la panacée du codeur !

Dart est Non-Nullable By Default (NNBD) ce qui signifie, qu'à part mention explicite, les variables sont par défaut non nullables. Pour passer en NNBD, il suffit de déclarer un **SDK** supérieur ou égal à 2.12 dans le **pubspec.yaml** de l'application et mettre à jour les dépendances à leur version null safe en consultant leur page sur https://pub.dev/ :

```
environment:
  sdk: ">=2.12.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  provider: ^5.0.0
  localstorage: ^4.0.0+1
```

À partir de cet upgrade, les variables nullables doivent être explicitement déclarées avec un **?** après leur type. Par exemple, **Produit? p** est nullable, mais pas **Rayon r**. L'analyseur de code statique fait le reste !

Voici quelques extraits de code qui ont changé. Dans **courses**, la valeur de retour des méthodes et leurs paramètres pouvaient être **null**, cela engendrait des tests à plusieurs niveaux :

```
class Rayon {
  factory Rayon.fromMap(Map<String,
dynamic> map) {
    if (map == null) return null;
    return Rayon(map['nom'] as String);
...
class Produit extends ChangeNotifier {

  Map<String, dynamic> toMap() =>
      {'nom': nom, 'rayon': rayon?.
toMap(), 'quantite': quantite, 'fait':
fait};

  factory Produit.fromMap(Map<String,
dynamic> map) {
    if (map == null) return null;
    return Produit(map['nom'] as String,
```

Dans **courses2**, le code précédent se simplifie grandement, puisque les paramètres des méthodes ainsi que leur valeur de retour sont déclarés non nullables :

```
class Rayon {
  factory Rayon.fromMap(Map<String, dynamic>
map) =>
      Rayon(map['nom'] as String);
...
class Produit extends ChangeNotifier {
  Map<String, dynamic> toMap() =>
      {'nom': nom, 'rayon': rayon.toMap(),
'quantite': quantite, 'fait': fait};

  factory Produit.fromMap(Map<String,
dynamic> map) => Produit(
      map['nom'] as String,
```

Quand un code NNBD peut se compiler sans erreur, c'est que l'analyse statique a **prouvé** qu'il est null safe. Pourtant, le lecteur averti trouve que le code précédent est incorrect ! En effet l'opérateur **[]** appliqué à **Map<K, V>** peut renvoyer **null** :

```
abstract class Map<K, V> {
  V? operator [](Object? key);
```

map['nom'] est donc nullable et c'est parce qu'il est promu **as String** qu'il devient non nullable. Si à l'exécution **map['nom']** est effectivement **null**, une exception **TypeError** est levée. Par exemple, cela peut arriver lors d'un **readAll()** si le storage externe a été corrompu. Comme avec toute promotion de type, il convient donc de protéger le cast par un bloc **try/on** :

```
Future<void> loadAll() async {
  try {
    importFromMap(await _storage.read());
  } on Error {
    debugPrint('Erreur de lecture. Fallback sur
les données intégrées.');
    importFromMap(await readFromAsset('courses'));
  }
}
```

Voici un autre exemple où une méthode prend un paramètre possiblement `null`. L'analyseur statique reconnaît qu'on ne déférence jamais `p` quand il vaut `null` :

```
/// Met à jour ou ajoute un produit
void ctrlMajProduit(Produit? p, Produit maj) {
  if (p == null) {
    _addSingleProduit(maj);
  } else {
    p.nom = maj.nom;
    p.rayon = maj.rayon;
  }
  _produits.sort((a, b) => a.rayon.nom.
compareTo(b.rayon.nom));
  notifyListeners();
  saveAll();
}
```

En résumé, pour une nouvelle application ou bibliothèque, NNBD est de rigueur. Pour une base de code existante et déjà éprouvée en exploitation, la question se pose. L'outil `dart --migrate` peut aider dans ce cas à migrer une large base de sources vers la null safety **[11]**.

PARAMÈTRES OPTIONNELS ET NULL SAFETY

Dart permet de définir des méthodes avec des paramètres optionnels. Avant la NNDB, un tel paramètre recevait la valeur `null` quand l'appelant l'omettait. Avec la NNBD, il faut soit le déclarer nullable en suffixant son type par `?`, soit pourvoir une valeur par défaut.

Dans le constructeur de `Produit`, `quantite` et `fait` sont des paramètres positionnels optionnels auxquels on affecte des valeurs par défaut :

```
Produit(this.nom, this.rayon, [this.
quantite = 0, this.fait = false]);
```

Pour un paramètre nommé non nullable, si on ne fournit pas de valeur par défaut, on doit le rendre `required` ce qui contraint l'appelant à passer une valeur non nulle :

```
void foo(String pos, {bool opt_def =
false, required int opt_req}){}

foo('Saint-Étienne', opt_req: 42);
```

2.2 Formater

Si **VSCode** et **AndroidStudio/IntelliJ** formatent les sources Dart et proposent du refactoring automatique pendant l'édition du code, on peut aussi le faire en ligne de commande. La règle est simple : un format unique qui permet aux équipes de dev d'en finir avec les interminables discussions autour du sujet. Le seul paramètre modifiable est le nombre de caractères maximum par ligne, 80 par défaut.

Pouvoir formater le code en ligne de commande permet de l'intégrer dans un script de vérification et/ou de modification automatique, par exemple un hook Git.

Pour faire un essai, entrons, dans le répertoire `courses2` :

```
$ flutter format -l 50 lib/main.dart
$ git diff
...
 void main() {
-  if (!kDebugMode) debugPrint =
(String? message, {int? wrapWidth}) {};
+  if (!kDebugMode)
+    debugPrint =
+        (String? message, {int?
wrapWidth}) {};
...
```

Nous voyons que l'outil a séparé la première ligne de la fonction `main()` en trois lignes différentes afin qu'aucune ne dépasse 50 caractères. Il n'y a pas de réécriture de code, certains éléments sont simplement déplacés.

C'est assez fâcheux, car maintenant le code enfreint la règle `curly_braces_in_flow_control_structures` qui incite à utiliser des accolades quand une structure de contrôle ne tient pas sur une ligne :

```
$ dart fix --dry-run lib/
Computing fixes in lib (dry run)...
1 proposed fix in 1 file.
main.dart
  curly_braces_in_flow_control_structures
• 1 fix
```

`dart fix` peut aussi corriger le code automatiquement :

```
$ dart fix --apply lib/
Applying fixes...
main.dart
  curly_braces_in_flow_control_structures
• 1 fix
1 fix made in 1 file.
$ git diff
```

```
...
 void main() {
-  if (!kDebugMode) debugPrint =
(String? message, {int? wrapWidth}) {};
+  if (!kDebugMode) {
+    debugPrint =
+        (String? message, {int?
wrapWidth}) {};
+  }
```

Et voilà ! En réalité, il est beaucoup plus efficace d'utiliser le bon éditeur de sources avec les plugins Dart et Flutter et d'appliquer les quick fixes proposés. Une utilisation possible de la **CLI** serait de formater du code Dart généré par un outil de plus haut niveau, comme par exemple un générateur de formulaires de saisie.

Pour annuler les changements : `git reset --hard`.

2.3 Analyser

Le langage Dart vient avec un outil d'analyse de code qui, bien configuré, produit des erreurs pour non-respect des guidelines ou détection de certains bugs potentiels **[12]**. On peut le voir comme un « super linter ».

Le fichier `analysis_options.yaml` d'un package renseigne sur les règles de vérification à appliquer à son code. Dans `courses2`, on a choisi d'appliquer les mêmes règles que Google utilise en interne **[13]**. Elles sont disponibles dans le package `pedantic` que l'on a donc rajouté dans les `dev_dependencies` de `pubspec.yaml` :

```
dev_dependencies:
  flutter_test:
    sdk: flutter
  pedantic: ^1.11.0
```

Le fichier `analysis_options.yaml` de `courses2` référence ce package avec la clause `include` :

```
include: package:pedantic/analysis_
options.yaml
```

Voici un extrait des règles du fichier Google **[14]** :

```
linter:
  rules:
    - always_declare_return_types
    - annotate_overrides
    - await_only_futures
    - omit_local_variable_types
    - prefer_single_quotes
    - unawaited_futures
    - unnecessary_new
```

Dans la branche `pedantic` de `courses2`, on s'est intentionnellement éloigné des guidelines Google. `flutter analyse` nous remet dans le doit chemin :

```
$ git checkout pedantic
$ flutter analyze
Analyzing courses2...
   info • The method toMap should have a return type
but doesn't • lib/modele.dart:22:3 •
          always_declare_return_types
   info • Annotate overridden members • lib/modele.
dart:26:10 • annotate_overrides
   info • Unnecessary new keyword • lib/modele.
dart:176:15 • unnecessary_new
   info • Omit type annotations for local variables
• lib/modele.dart:183:5 • omit_local_variable_types
   info • Only use double quotes for strings
containing single quotes • lib/modele.dart:223:18 •
          prefer_single_quotes
   info • The declaration '_pedantic' isn't
referenced • lib/modele.dart:233:15 • unused_element
   info • 'await' applied to 'int', which is not
a 'Future' • lib/modele.dart:235:5 • await_only_
futures
   info • `Future` results in `async` function
bodies must be `await`ed or marked `unawaited` using
          `package:pedantic` • lib/modele.dart:238:5
• unawaited_futures
8 issues found. (ran in 1.0s)
```

Certaines issues peuvent être corrigées par `dart fix` :

```
$ dart fix --apply
...
6 fixes made in 1 file.
$ flutter analyze
...
2 issues found. (ran in 1.0s)
$ git diff
+  @override
   String toString() => 'Rayon(nom: $nom)';
-      rayon = new Rayon(nom);
+      rayon = Rayon(nom);
-    Rayon rayon = _addSingleRayon(produit.
rayon.nom);
+    var rayon = _addSingleRayon(produit.
rayon.nom);
-      debugPrint("Erreur de lecture.
Fallback sur les données intégrées.");
```

```
+      debugPrint('Erreur de lecture.
Fallback sur les données intégrées.');
-    await _pedantic0();
+    _pedantic0();
-    saveAll();
+    await saveAll();
```

Dans un éditeur de sources, le plug-in Dart se charge de lancer l'analyse en background et le développeur est incité à corriger les problèmes avant son commit en utilisant les fixes automatiques.

Pour annuler les changements : **`git reset --hard && git checkout master`**.

2.4 Documenter

C/C++ a **Doxygen**, **Python** a **Sphinx**, Dart a son **dartdoc [15]**. On le télécharge, compile et installe avec **`pub`** :

```
$ flutter pub global activate dartdoc
Warning: Pub installs executables into $HOME/
travail/flutter/.pub-cache/bin, which is not
on your path.
You can fix that by adding this to your
shell's config file (.bashrc, .bash_profile,
etc.):

  export PATH="$PATH":"$HOME/travail/flutter/.
pub-cache/bin"

Activated dartdoc 0.42.0.
```

On réalise l'**`export`** suggéré et on l'exécute dans **`courses2`** :

```
$ dartdoc
Documenting courses2...
...
Validating docs...
Found 1 warning and 0 errors.
Documented 5 public libraries in 2.5 seconds
Success! Docs generated into [...]/courses2/
doc/api
```

Il suffit alors de lancer un serveur web **`python3 -m http.server 8000 --directory doc/api`** et d'ouvrir http://localhost:8000/ depuis un navigateur (figure 6).

Les commentaires de documentation sont introduits dans le source par **`///`**. La syntaxe n'est autre que du **Markdown**. On peut référencer des noms de paramètres ou tout autre identifiant avec **`[]`**. Le fichier **`README.md`** est inséré dans la page d'accueil de la documentation. Pour documenter une bibliothèque, on insère le mot clé **`library`** au début de fichier. Seuls les membres publics sont documentés. Les membres commençant par **`_`** sont donc ignorés. Les membres disposant d'un getter, mais pas de setter sont marqués read-only. Les relations d'héritage entre classes sont documentées dans un fil d'Ariane. **`dartdoc`** génère des liens externes pour les classes Flutter et Dart référencées par des membres publics du package.

Exemple :

```
/// Cette bibliothèque définit les classes du
modèle de données de l'application
/// courses2 : [Rayon], [Produit] et
[VueModele].
library modele;

/// Un produit défini par son [nom] et son
[rayon].
class Produit extends ChangeNotifier {
  /// Le [nom] de ce [Produit].
  String nom;

  /// Crée un nouveau Produit avec son [nom] et
son [rayon]. Par défaut, la
  /// [quantite] est initialisée à `0` et [fait]
à `false`.
  Produit(this.nom, this.rayon, [this.quantite =
0, this.fait = false]);

/// Le vue modèle et son contrôleur.
class VueModele extends ChangeNotifier {
  final StorageStrategy _storage;

  late Future<void> _isLoaded;

  /// [isLoaded] se réalise quand le [loadAll()]
initial est terminé.
  Future<void> get isLoaded => _isLoaded;

  final Rayon _divers = Rayon('Divers');

  /// Le [Rayon] 'Divers'.
  Rayon get divers => _divers;

  /// Crée le modèle et charge les données
suivant la [StorageStrategy] en
  /// paramètre.
  VueModele(this._storage) {
    _isLoaded = loadAll();
  }

  /// Incrémente la quantité du [Produit] [p].
  void ctrlProduitPlus(Produit p) {
```

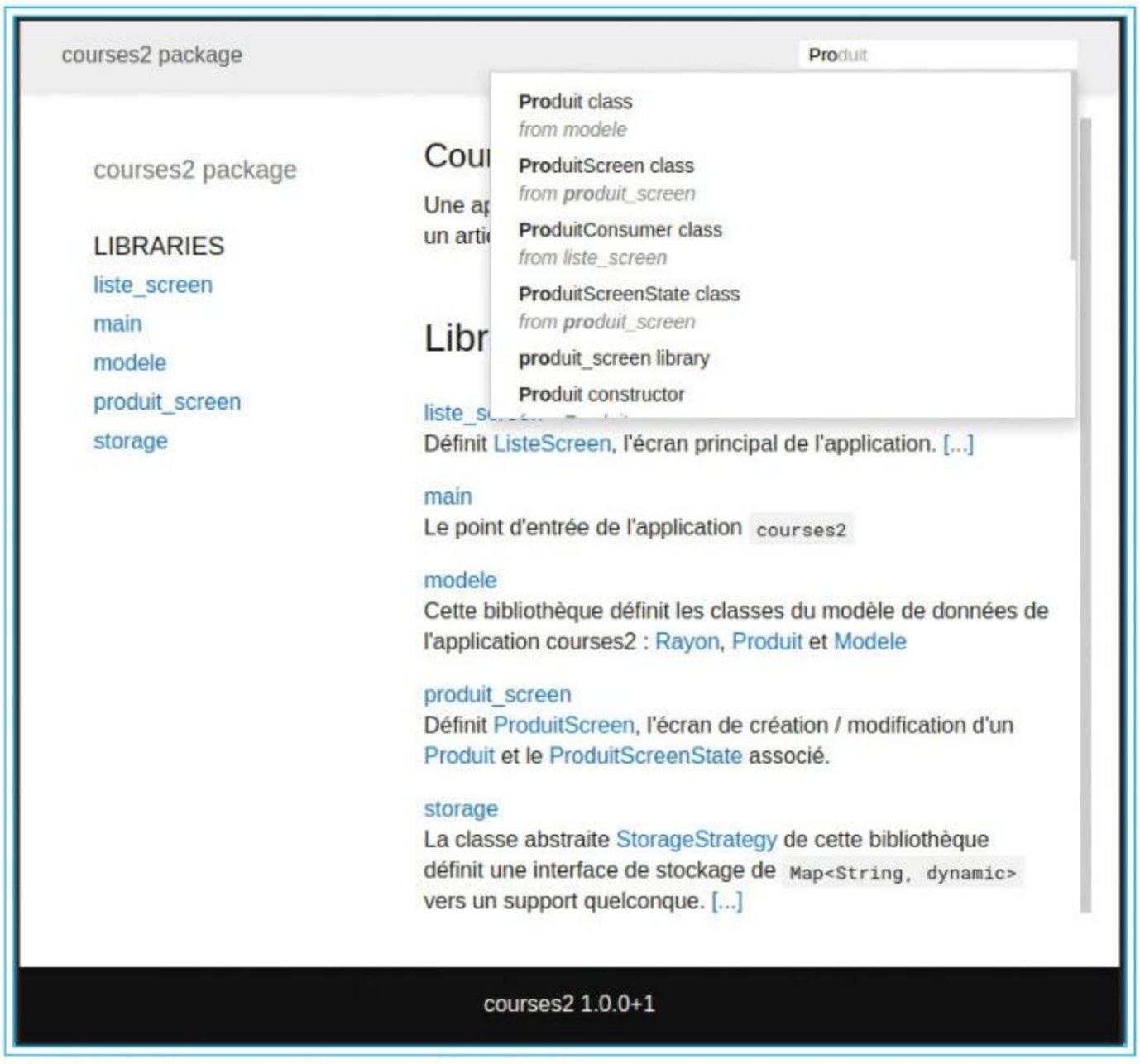


Fig. 6 : La documentation servie en local.

Un guide plus avancé, définissant les conventions d'écriture de la documentation, est disponible sur https://dart.dev/guides/language/effective-dart/documentation.

2.5 Tester

Flutter propose trois catégories de tests automatiques **[16]** : les tests unitaires d'API, les tests de Widgets et les tests d'intégration. Cet article ne couvre que les deux premières catégories. On trouvera de la documentation sur la troisième sur https://flutter.dev/docs/cookbook/testing/integration/introduction.

Dans **courses2**, on a deux tests unitaires sur le « Vue Modèle ». Ils sont dans **tests/modele_test.dart** et contiennent « modele » dans leur nom. On peut les exécuter en CLI :

```
$ flutter test -r expanded --plain-name="modele"
00:00 +0: [...]/modele_test.dart: modele: init
00:00 +1: [...]/modele_test.dart: modele: plus, moins, selection
00:00 +2: All tests passed!
```

Le code du test **init** est assez explicite : On crée un **VueModele** en mémoire avec des données connues, **dataset1**, puis on décrit le test par une suite d'assertions. Le test passe si toutes les assertions sont vraies.

```
import 'package:flutter_test/flutter_test.dart';
...
void main() async {
  modele = VueModele(MemoryMapStrategy(dataset1));
  await modele.isLoaded;
  test('modele: init', () {
    assert(modele.produits.length >= 4);
    assert(modele.produits[0].nom == 'Escalope de porc');
    assert(modele.produits[3].nom == 'Sel');
  });
```

Si ces tests sont utiles au développeur qui peut vérifier en local la solidité de son code, on peut imaginer les exécuter à chaque commit sur une plateforme d'intégration continue.

Les tests de widgets ne sont guère plus compliqués à écrire. On les introduits par **testWidgets** et on utilise **expect** pour vérifier que certains widgets sont présents ou pas, qu'ils contiennent bien le texte ou l'icône attendus. Les widgets ne sont pas affichés, ils sont juste buildés en mémoire. Un objet **WidgetTester** permet d'actionner la « pompe à frames », c'est-à-dire d'exécuter des cycles Flutter sur demande qui font changer d'état l'arbre des widgets. De petites subtilités existent toutefois, notamment lors de l'utilisation de **FutureBuilder** et de méthodes asynchrones. Les commentaires de **widget_test.dart** fournissent quelques informations à ce sujet.

Par exemple, le code suivant simule un tap sur l'icône ***(+)***, **add_circle**, du premier élément de la liste des produits, vérifie que la quantité du produit a changé dans le modèle, puis actionne la pompe et vérifie que la nouvelle valeur est mise à jour dans le widget :

```
testWidgets('ListeScreen Widget Test', (WidgetTester
tester) async {
  await tester.runAsync(() async {
    await tester.pumpWidget(MaterialApp(home:
ListeScreen()));
  });
  ...
  var icon = find.byIcon(Icons.add_circle);
  await tester.tap(icon.first);
  // Le modèle a été mis à jour mais pas encore l'UI
  assert(modele.produits[0].quantite == 1);
  expect(find.text('1'), findsNothing);
  // On prend une autre image
  await tester.pump();
  // Le widget doit afficher la nouvelle valeur
  expect(find.text('1'), findsOneWidget);
});
```

Comme on peut exécuter les tests de widgets en CLI sans besoin d'un terminal graphique, on peut aussi les inclure sur une plateforme CI/CD :

```
$ flutter test -r expanded --plain-
name=ListeScreen
00:00 +0: [...]/widget_test.dart:
ListeScreen Widget Test
00:00 +1: All tests passed!
```

CONCLUSION

Certains sujets auraient mérité d'être abordés dans cet article, mais tout à une fin ! Je pense aux outils de mesure de performance et d'optimisation, à **dart:ffi** qui permet d'appeler des API C natives, du projet « FlutterFire » qui tente d'harmoniser l'accès à **FireBase** depuis les targets mobiles, Web et desktop.

Mais il faut déjà se concentrer sur l'évolution de l'application « Courses » vers le monde du connecté et du cloud. Suspense... ■

RÉFÉRENCES

[1] A. BASTY, « Flutter : Applications mobiles, web et desktop », GNU/Linux Magazine N°248, mai 2021 : https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-248/Flutter-applications-mobiles-web-et-desktop

[2] GOOGLE, « Understanding null safety » : https://dart.dev/null-safety/understanding-null-safety

[3] Wikipédia, « Intégration continue » : https://fr.wikipedia.org/wiki/Int%C3%A9gration_continue

[4] Wikipédia, « Modèle-vue-vue modèle » : https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-vue_mod%C3%A8le

[5] Wikipédia, « Stratégie (patron de conception) » : https://fr.wikipedia.org/wiki/Strat%C3%A9gie_(patron_de_conception)

[6] Wikipédia, « Representational state transfer » : https://fr.wikipedia.org/wiki/Representational_state_transfer

[7] Google, « List of state management approaches » : https://flutter.dev/docs/development/data-and-backend/state-mgmt/options

[8] Google, « Simple app state management » : https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple

[9] Wikipédia, « Observateur (patron de conception) » : https://fr.wikipedia.org/wiki/Observateur_(patron_de_conception)

[10] Google, « Sound null safety » : https://dart.dev/null-safety

[11] Google, « Migrating to null safety » : https://dart.dev/null-safety/migration-guide

[12] Google, « Customizing static analysis » : https://dart.dev/guides/language/analysis-options

[13] Google, « pedantic 1.11.0 » : https://pub.dev/packages/pedantic

[14] Google, « analysis_options.1.11.0.yaml » : https://github.com/google/pedantic/blob/master/lib/analysis_options.1.11.0.yaml

[15] Google, « dartdoc » : https://github.com/dart-lang/dartdoc

[16] Google, « Testing » : https://flutter.dev/docs/cookbook/testing

Chez votre marchand de journaux

et sur **www.ed-diamond.com**

PAPIER
en kiosque

FLIPBOOK HTML5
sur www.ed-diamond.com

CONNECT
LA DOCUMENTATION TECHNIQUE DES PROS DE L'IT
sur connect.ed-diamond.com

PAS DE BRAS, MAIS QUAND MÊME DU CHOCOLAT : L'ÈRE DES ASSISTANTS VOCAUX

DAVID BLASKOW
[Artisan du mal et responsable du mauvais goût technologique mondial pour Data Essential]

MOTS-CLÉS : SIRI, ALEXA, GOOGLE HOME, INTERFACE UTILISATEUR VOCALE, GOLANG

Nos interactions avec les ordinateurs (et autres smartphones) sont dignes du Moyen Âge. Qu'est-ce qui a réellement changé en la matière, ces 50 dernières années ? Rien : nous avons juste remplacé notre bon vieux clavier mécanique par un clavier virtuel, la belle affaire ! Pourtant, nous pouvons faire mieux, et nous devons faire mieux : bienvenue dans l'ère des assistants vocaux virtuels ! Nous allons découvrir ensemble ce qu'ils sont, et créer notre première application vocale.

De **HAL 9000** à **Jarvis**, les IA (pour Intelligences Artificielles) avec lesquelles nous interagissons par la parole ont toujours déchaîné l'imagination. Mais bien loin de n'être que de simples fictions, ces IA existent bel et bien et se nomment **Siri**, **Alexa** ou **Google Home**, pour ne citer que les plus célèbres. Elles sont à portée de main, si proches que n'importe qui peut créer sa propre application en moins d'une demi-heure ! Et c'est ce que nous allons faire : donner vie à un monstre de cuivre et de silicium qui va peut-être causer notre perte ! Mais d'abord, nous allons un peu casser le mythe en découvrant le vrai visage de ces assistants vocaux.

1. DE HOLLYWOOD À LA SILICON VALLEY

1.1 IA ou simple bot ?

Qu'est-ce qu'Alexa en définitive ? Je vais être méchant avec elle : une simple interface utilisateur. En effet, c'est une user interface (UI) et ce qui la distingue des autres UI, c'est simplement qu'elle est vocale (Vocal User Interface ou VUI). Donc, ce n'est pas aussi magique que nous pourrions nous le représenter de prime abord.

Ce qui est plus intéressant, c'est de comprendre comment ça fonctionne. Et rien de tel que de prendre un exemple : « Alexa, je voudrais écouter du jazz ».

Première étape, à l'appel de son nom, Alexa se met en écoute active, c'est-à-dire qu'elle s'apprête à réagir à ce que nous allons lui dire. Oui, elle écoute en permanence, tout comme nos téléphones, nos tablettes et nos ordinateurs... mais aussi nos frigos, nos télévisions, et en définitive tout appareil connecté disposant d'un micro. Si Alexa vous rend paranoïaque, car nous nous souvenons tous de cette anecdote où Alexa a appelé d'elle-même la police alors qu'un homme battait sa petite amie **[1]**, vous devriez l'être tout autant avec tous vos autres gadgets.

Mais oublions cette anecdote un instant, et passons au moment où les choses sérieuses commencent, pour ne pas dire la magie : la phrase « je voudrais écouter du jazz » va être traitée par des algorithmes de Machine Learning (ML) : c'est ce qui s'appelle le Natural Language Processing (NLP). Plus concrètement, qu'est-ce que ça signifie ? Qu'Alexa est en mesure de comprendre l'intention que vous exprimez. Ici, l'intention est d'écouter de la musique, mais plus précisément de la musique jazz. Le jazz n'est pas ici votre intention, nous sommes bien d'accord, c'est bien d'écouter de la musique. Et tout ceci va être traduit en une requête **JSON** qu'Alexa va envoyer à une API ou à un middleware, plus communément appelé un endpoint dans le jargon d'**Alexa Skills**, ce qui permet de laisser planer le doute sur ce que c'est vraiment.

Qui a-t-il de magique dans tout ça ? Pas mal de choses à mes yeux. Vous vous souvenez de ces vieilles messageries vocales avec lesquelles nous interagissions si pauvrement ? Si « dites : je veux être mis en relation avec un opérateur pour être mis en relation avec un opérateur » vous rappelle quelque chose, alors vous voyez de quoi je parle. Pas de ça, ici. Nous avons formulé notre demande de la sorte : « Alexa, je voudrais écouter du jazz », mais nous aurions tout aussi bien pu dire :

« Alexa, passe-moi du jazz. »

« Alexa, j'ai envie d'écouter de la musique jazz. »

« Alexa, joue du jazz. »

Pas de phrases stéréotypées, Alexa se charge de comprendre votre « langage naturel ». L'autre point intéressant, c'est cette détection d'intention. Alexa ne réagira pas de la même manière à « je veux écouter du jazz » et à « je veux manger un plat végétarien », et heureusement, me direz-vous. Du coup, notre API aura beau jeu de faire le tri et d'appeler la bonne fonction pour réagir à telle ou telle demande.

Une fois la partie NLP franchie, Alexa génère une magnifique (hum hum) requête en JSON qu'elle transmet à votre API. Charge à votre application d'y répondre correctement. Dans notre cas, et si je continue avec notre exemple, puisqu'il s'agit d'écouter de la musique, Alexa va très certainement regarder ce qu'**Amazon Music** a à proposer (oui, je sais, Amazon est controversé, mais je l'utilise juste en guise d'introduction aux assistants vocaux, car c'est le plus abouti ; cependant, une fois familiarisé avec la façon de faire des VUI, je vous invite à en découvrir d'autres, comme Google Home ou mieux encore, si vous êtes un fan inconditionnel de **FOSS**, un **Nabaztag [2]** avec **Mycroft [3]** ou **Kalliope [4]**). Sans doute qu'une station musicale est disponible, et Alexa va finir sa séquence par un « j'ai trouvé la station J'aime le Jazz sur Amazon Music », et la diffusion pourra commencer.

Si nous nous résumons un peu, Alexa est donc une IA (merci au NLP), mais qui agit de la même manière que n'importe quelle autre interface utilisateur. Alors, en quoi est-ce utile ? D'une part, c'est en moyenne 5 fois plus rapide que d'utiliser toute autre interface. Je suis sûr d'avoir ma station de jazz bien plus vite en le demandant à Alexa qu'en allant sur **Spotify** dans mon navigateur pour rechercher par moi-même cette même station. Et d'autre part, c'est une interface on ne peut plus utile lorsque nous sommes occupés à faire autre chose de nos mains, comme lorsque nous conduisons ou nous faisons la cuisine (ah, rien de tel que d'essayer de faire défiler une recette avec les mains pleines de farine !).

1.2 Alexa est bien loin d'être la version féminine de Jarvis...

Nous venons de le voir, Alexa pourrait être décrite comme un bot vocal, ce qui est cool, mais ça ne va pas sans certaines limitations. En effet, les études **[5]** montrent que cette technologie est encore relativement immature.

Premier point, si ce que vous dites est bien transcrit par les VUI, l'interprétation qui en est faite laisse parfois à désirer. Par exemple, si je dis : « Alexa, passe-moi du jazz », il y a des chances pour qu'elle me dise qu'elle ne peut pas me mettre en relation avec jazz, car elle ne l'a pas trouvé dans mes contacts. « Passe-moi » peut être interprété comme « téléphone à », et ce n'est que le contexte qui permet de bien comprendre l'intention réelle. Ce n'est déjà pas toujours simple pour un humain, alors une machine, pensez donc !

Ensuite, les VUI sont surtout efficaces pour réaliser des actions simples : « Alexa, qui est le réalisateur de 2001 l'Odyssée de l'espace » vous retournera Stanley Kubrick, par contre, « Alexa, quel acteur a joué dans L'Armée des 12 singes et dans Fight Club ? » a peu de chances de trouver une réponse. Cependant, petit bémol, ce n'est pas un trait caractéristique des assistants vocaux, mais de tous les super bots. Grâce au NLP, Alexa enverra la bonne requête : l'intention est de trouver un acteur, et cet acteur a joué dans les deux films suivants, mais l'API qui est ensuite appelée devrait être en mesure de faire deux appels à l'API d'**IMDb** pour récupérer la liste des acteurs de ces deux films, puis de les comparer, et enfin de retourner le nom de celui qui apparaît dans les deux listes. Pas impossible, mais compliqué.

Enfin l'intégration avec vos informations personnelles. Vous pouvez interagir avec votre boîte mail ou votre calendrier, mais c'est à peu près tout. D'un autre côté, ce n'est pas plus mal non plus, car les assistants vocaux ne sont pas vraiment sûrs : n'importe qui peut parler à mon Alexa, et de là faire des actions malicieuses. Donc, il y a peu de chance pour le moment que j'ai envie de l'intégrer avec mon application bancaire.

Ainsi, il ne faut pas se voiler la face, il y a bel et bien des limitations dans ce que nous pouvons faire avec les assistants vocaux. Mais le potentiel est bel et bien là, prêt à être exploité !

1.3 ...Mais c'est l'avenir !

Pour beaucoup, c'est encore de la science-fiction, pourtant qui ne rêverait pas d'avoir un majordome comme Jarvis ? Beaucoup de choses sont d'ores et déjà possibles, et je vous invite à vous laisser entraîner par votre imagination. Voilà par exemple comment pourrait se passer un matin classique :

« Alexa, bonjour. »

« Bonjour, aujourd'hui nous sommes le 4 janvier, et oh ! C'est la journée mondiale des bots. Sait-on jamais, des fois que vous auriez envie de me faire un petit cadeau... »

« Alexa, quel temps fait-il dehors ? »

« Actuellement, il fait 1 degré et des risques de chute de neige sont prévus dans la journée. »

« Alexa, allume la lumière de la chambre. »

« Alexa, prépare mon café. »

Notez qu'ici, ça suppose bien évidemment d'avoir des ampoules connectées, et d'avoir identifié dans Alexa chaque pièce de la maison, mais c'est faisable. Autre point intéressant, quand nous parlions des limites, voici une chose qu'Alexa ne peut pas faire : comprendre deux intentions. Allume la lumière et prépare-moi mon café auraient pu naturellement faire partie de la même demande, mais oups, ça ne fonctionne pas comme ça. À l'heure actuelle, nous sommes obligés de faire deux demandes bien distinctes.

« Alexa, est-ce que j'ai des messages ? »

« Vous avez reçu un e-mail de votre mère. Voulez-vous que je vous le lise ? »

« Oui, s'il te plaît. »

La formule de politesse est totalement optionnelle, Alexa ne se vexe jamais, même quand nous sommes un peu rudes avec elle. Mais, croyez-moi, il est facile de se prendre au jeu et d'interagir avec elle, comme s'il s'agissait d'une vraie personne.

Enfin voilà, vous avez saisi le concept. Vous lui demanderez ensuite de vous donner les infos pendant que vous prenez votre petit déjeuner, à moins que vous ne préfériez écouter un podcast ou de la musique. Il y a plein de menus services qu'Alexa est capable de vous rendre (encore une

fois, je dis Alexa pour ne pas dire assistant vocal, parce que c'est plus simple ; mais rassurez-vous, je ne détiens aucune action chez Amazon).

2. RENCONTRE DU 4E TYPE

2.1 Êtes-vous prêt ?

Je suis sûr que vous l'êtes, et si les premières marches ne sont pas très hautes, elles peuvent tout de même être un peu intimidantes. Mais allez sans crainte, pour bien comprendre l'outil, il est nécessaire d'expérimenter un peu.

Première étape, rendez-vous sur https://developer.amazon.com/fr-FR/alexa/ pour créer un compte utilisateur. C'est gratuit et vous pouvez aller assez loin sans avoir à débourser le moindre centime. Une fois votre compte créé, rendez-vous sur la console : https://developer.amazon.com/alexa/console/ask (figure 1). C'est ici que nous allons créer notre première Skill.

Donnez-lui le nom que vous voulez, après tout il ne s'agit ici que de s'entraîner, et choisissez votre langue et le type custom (figure 2, page suivante).

Alors, un homme averti en vaut deux : c'est tentant de prendre Français (FR), et c'est ce que nous allons d'ailleurs faire dans cet article. Mais la langue française étant un peu plus... complexe que la langue anglaise, vous serez peut-être parfois surpris par la couche de ML. Qu'à cela ne tienne, ce n'est qu'un obstacle, et il est loin d'être insurmontable. Pour l'anecdote, j'ai essayé de faire une application en anglais, et j'ai ri comme jamais en me rendant compte que mon accent était si épouvantable qu'Alexa ne comprenait rien à ce que je lui disais. Alors s'il vous prend l'envie de piquer un fou rire, je vous conseille cette option, à moins bien sûr que vous ne soyez parfaitement anglophone.

Nous allons créer notre Skill from scratch (figure 3, page suivante), enfin donc depuis zéro, et pour cela nous devons valider un captcha.

Ça met un peu de temps à se lancer, l'occasion d'aller prendre un café (figure 4, page 25). Comme quoi la technologie évolue, mais les tutos ne changent pas, nous sommes toujours obligés de boire souvent du café.

Maintenant que notre Skill est prête, nous sommes redirigés sur la page de la figure 5 (voir page 25).

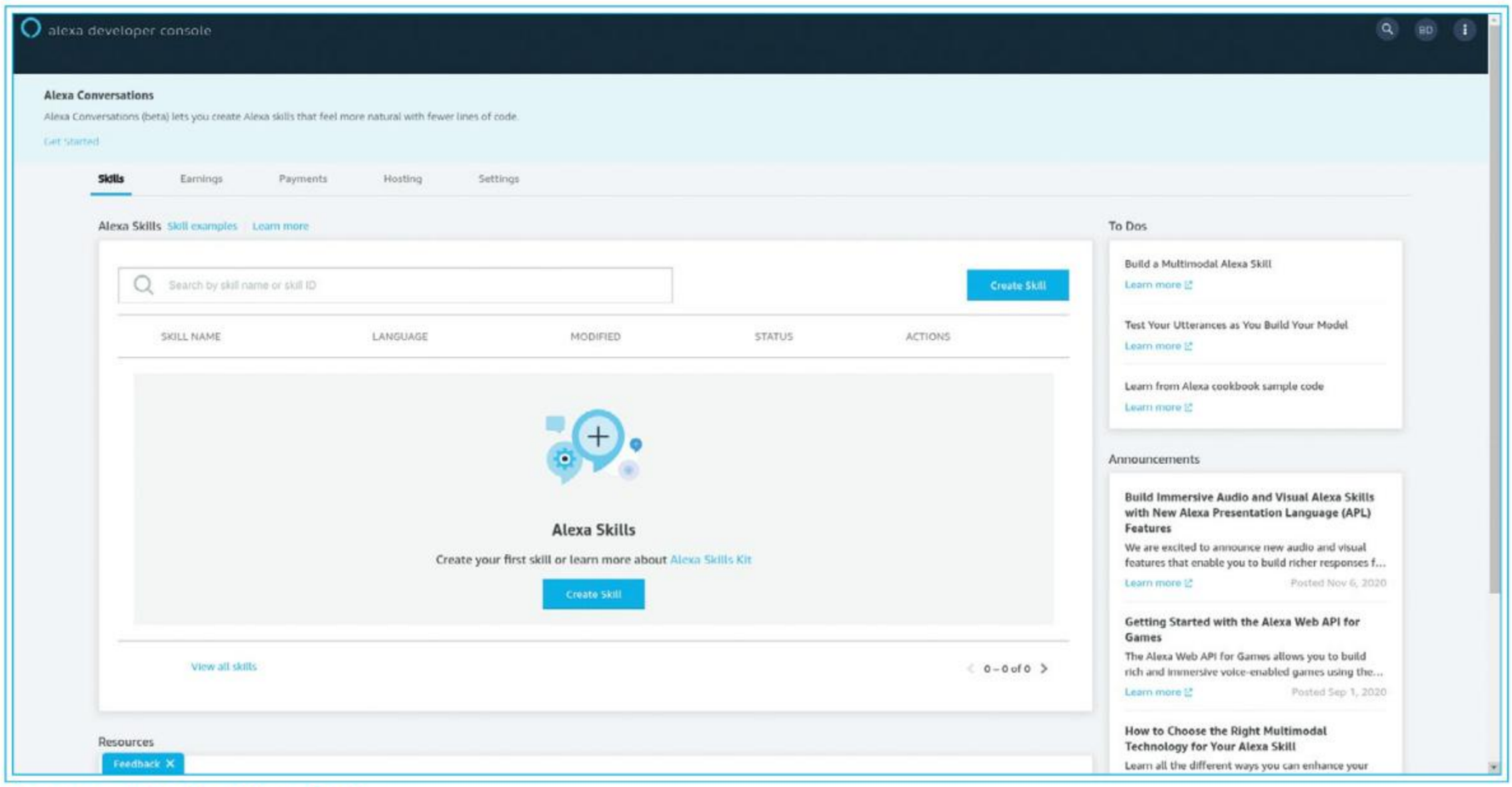


Fig. 1 : La console d'Alexa Skills.

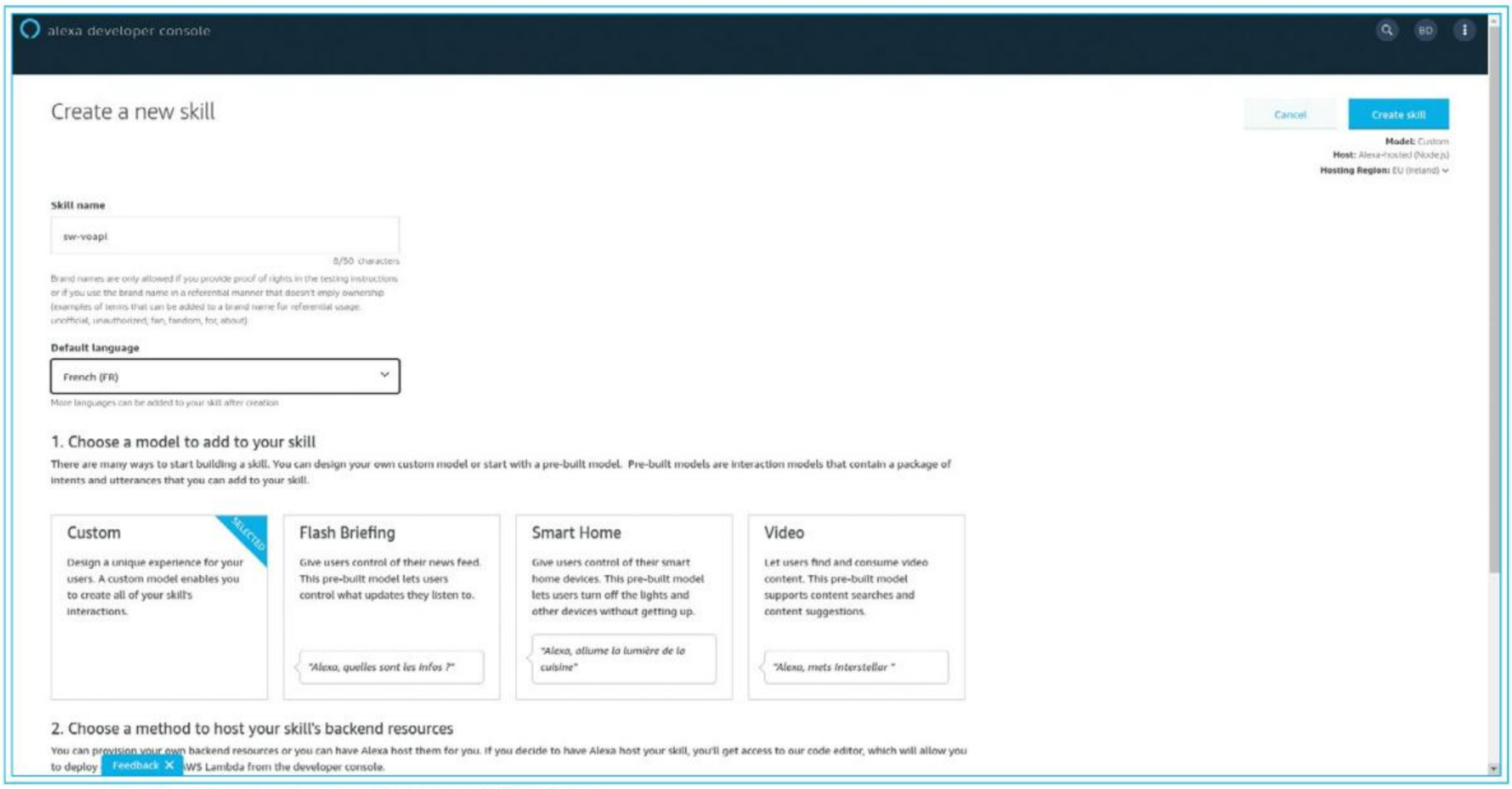


Fig. 2 : Donnez un nom à votre application.

2.2 Invocation

L'invocation est très importante, car c'est en réalité le nom de notre application. Qu'est-ce que ça signifie ? Imaginez que vous avez créé une application qui donne des recettes de cuisine à base de champignons. Vous pourriez vous imaginer pouvoir demander à Alexa : « Alexa, quelle soupe pourrais-je faire avec des cèpes ? » ; c'est loin d'être un raisonnement absurde, mais ça ne marche pas comme ça. En réalité, vous devez invoquer votre application par son nom dans un premier temps :

« Alexa, lance champignons merveilleux ».

Si cette application existe, Alexa va passer la main à votre Skill, qui à son tour pourra donner son message de bienvenue :

« Bienvenue dans l'application champignons merveilleux ! Quel champignon voulez-vous cuisiner ? »

« Je voudrais faire une soupe à base de cèpes. »

Et là, la magie opérera.

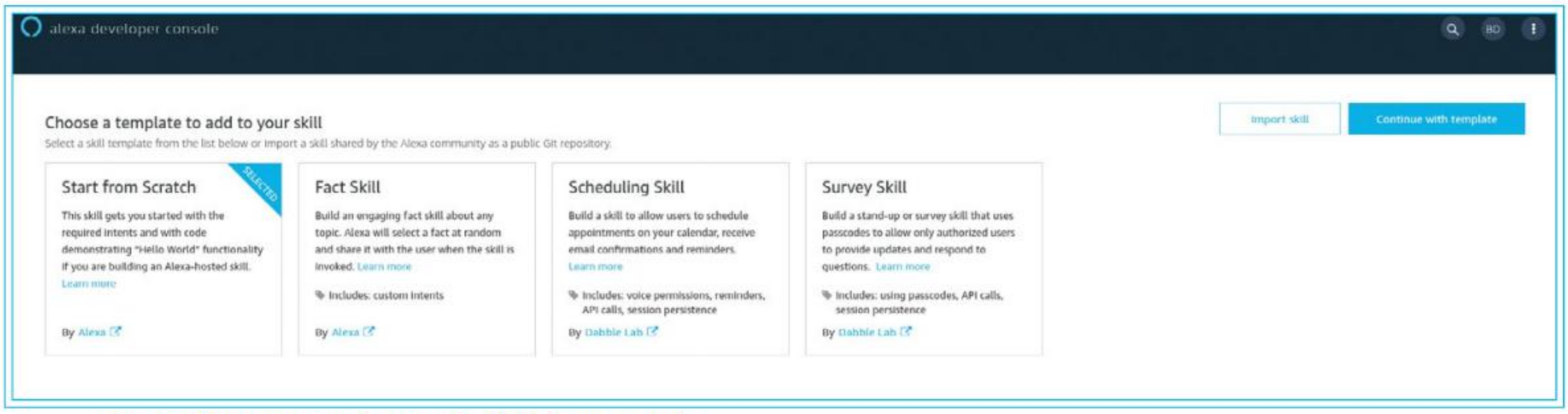


Fig. 3 : Nous prenons le type de Skill from scratch.

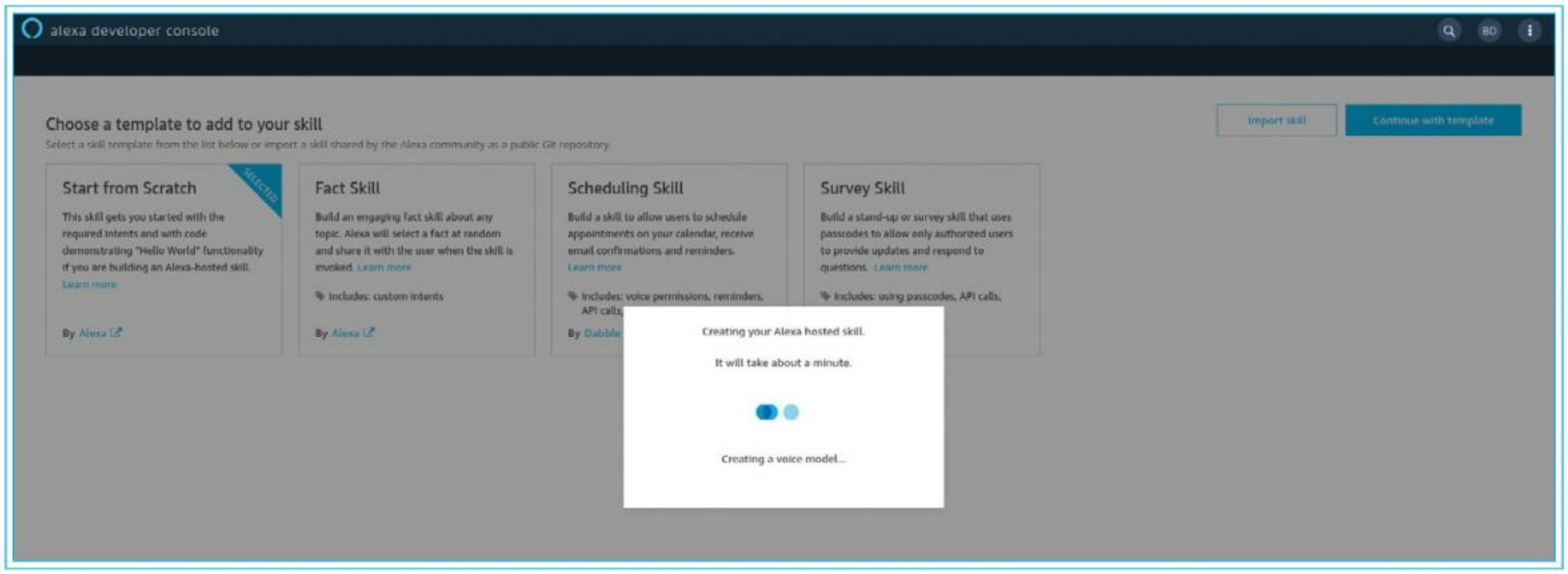


Fig. 4 : Que ce soit avec Gmail, le cloud ou Alexa, il faut toujours attendre… Je ne sais pas vous, mais moi ça me rappelle ces bonnes vieilles pubs pour l'ADSL 512K de Cegetel : la vitesse vous manque **[6]** ?

Remplacez change me par le nom de votre application (figure 6, page suivante). Nous allons utiliser « Holo Disque ». Les puristes de **Stars Wars** vont m'en vouloir de ne pas dire **Holodisc [7]**, mais laissez-moi vous expliquer. Le nom utilisé pour l'invocation doit être composé de deux mots au minimum : donc « Holo » et « Disque » dans mon cas. Et puis honnêtement, cela vous choque-t-il parce que c'est écrit en français ? N'oubliez pas que nous créons une application vocale, alors quelle importance ?

2.3 Slot

Pour ceux qui sont familiers des regexp, un slot c'est une sorte de regexp aux amphétamines : c'est un mot ou une suite de mots qu'Alexa est en mesure de reconnaître dans une phrase, un peu comme un motif. Par exemple, je peux identifier Pierre, Paul et Jacques comme étant des prénoms. Ou fromage, champignons et tomate comme des aliments. C'est ici que se trouvent les modèles de Machine Learning.

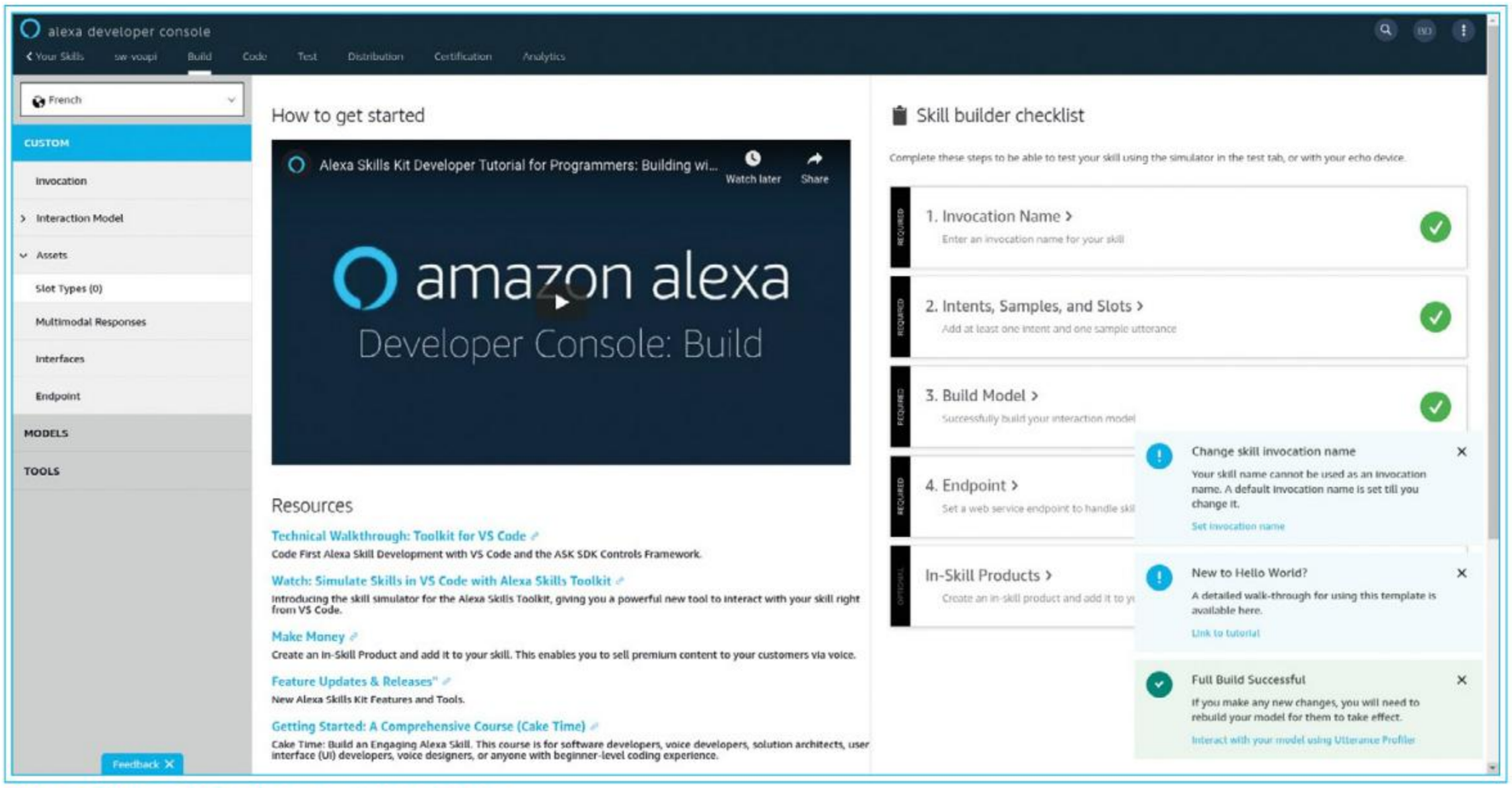


Fig. 5 : Page principale de la console d'Alexa.

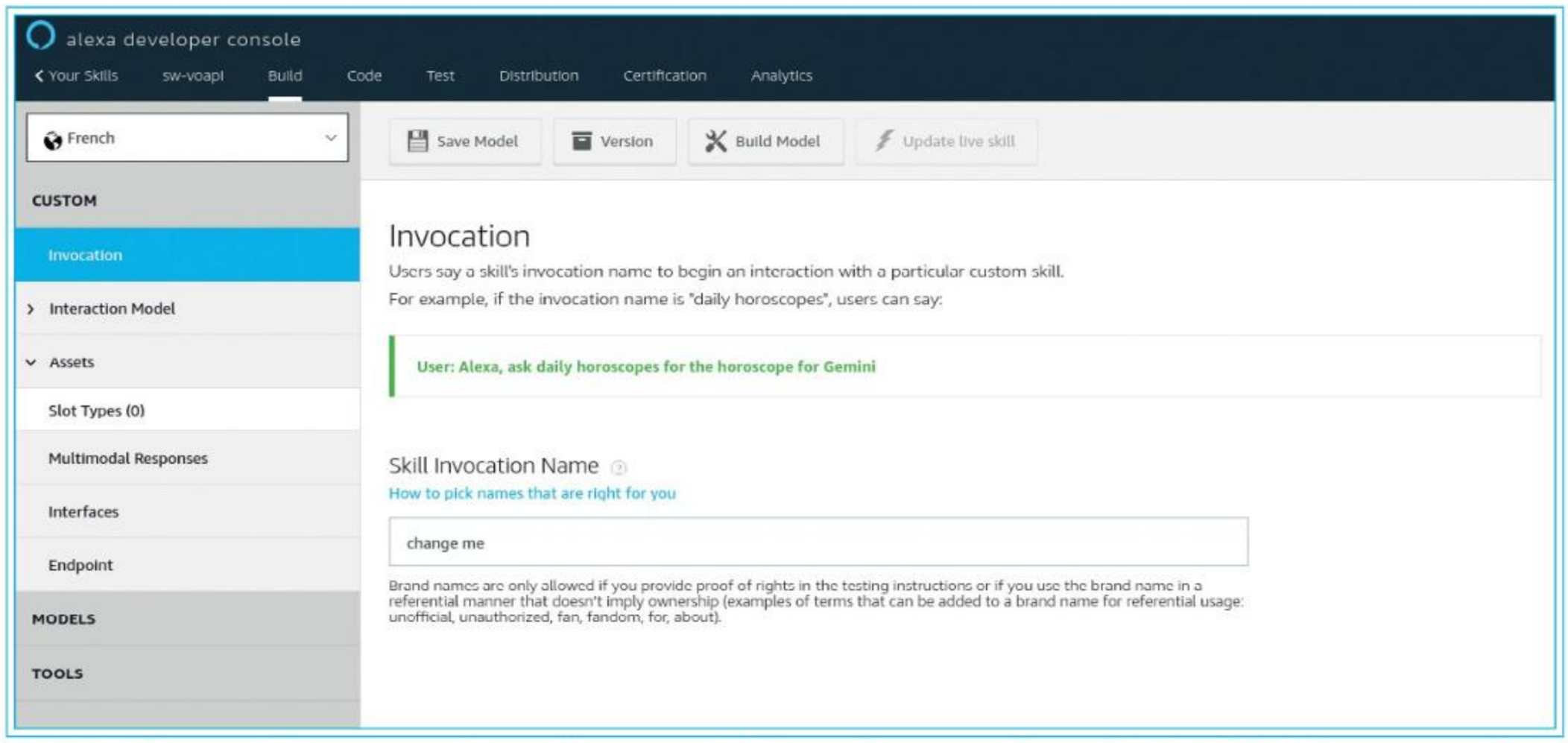


Fig. 6 : Le fameux « change me » qui a aussi longtemps été le mot de passe root des applications Sun Microsystems… Sans doute l'œuvre d'un transfuge.

Disons que nous écrivons une application pour commander une pizza. Alexa peut par exemple demander : « Quels ingrédients voulez-vous sur votre pizza ? », et vous lui répondrez « Tomates, fromage et champignons ». Puis un jour, vous lui direz : « Olives, ananas et fromage ». Oups, à ce stade, Alexa ne sait pas, pas plus que moi d'ailleurs, que l'ananas peut être un ingrédient pour une pizza. Mais pas d'inquiétude, vous en êtes notifié, et vous pourrez alors améliorer votre modèle.

Pour notre application Holo Disque, notre premier slot (figure 7) correspond au nom des personnages.

Je l'ai appelé très prosaïquement **`sltype_sw_people_name`**. J'y déclare trois attributs :

- un nom ;
- un ID, afin de pouvoir discuter simplement avec l'API **`swapi.dev`** sans ajouter un niveau de cache ;

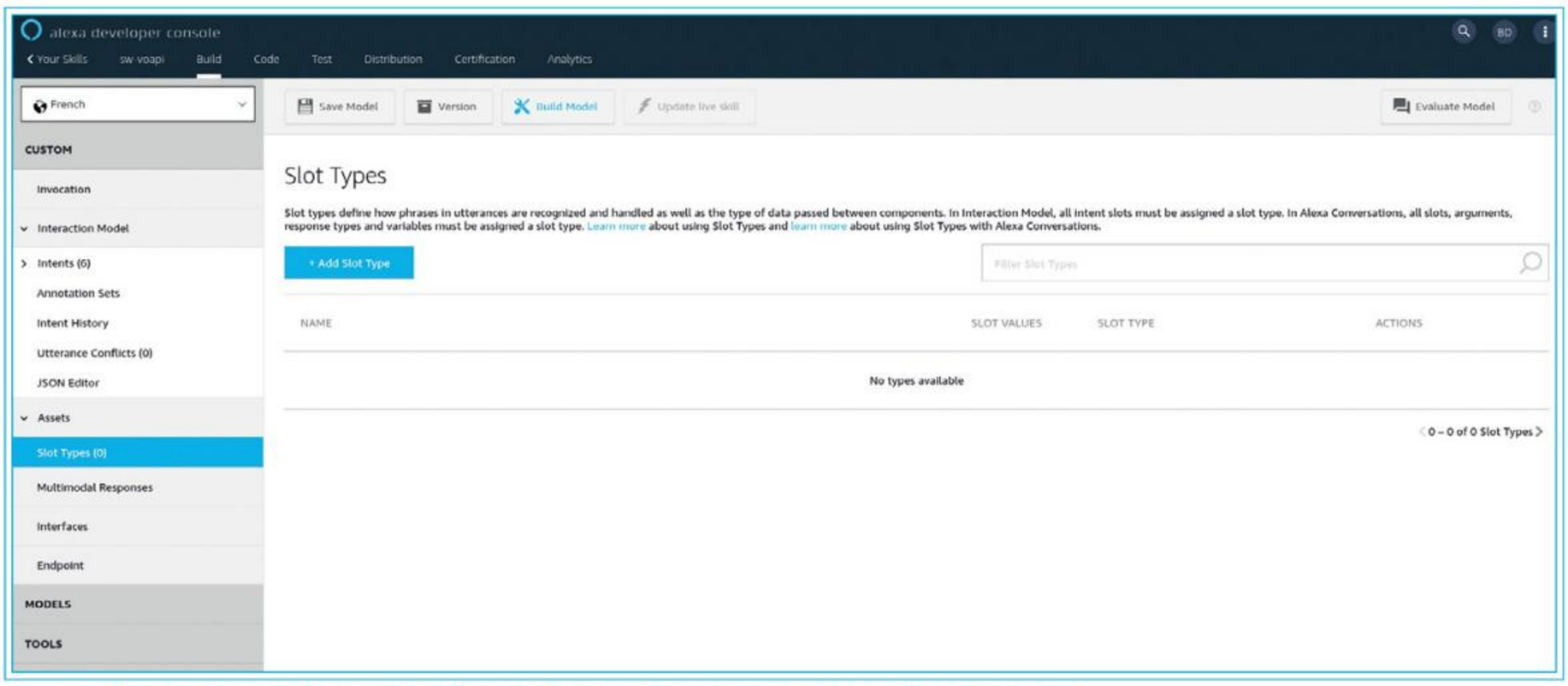


Fig. 7 : Notre tout premier slot, mais il y a fort à parier que ce ne soit pas le dernier.

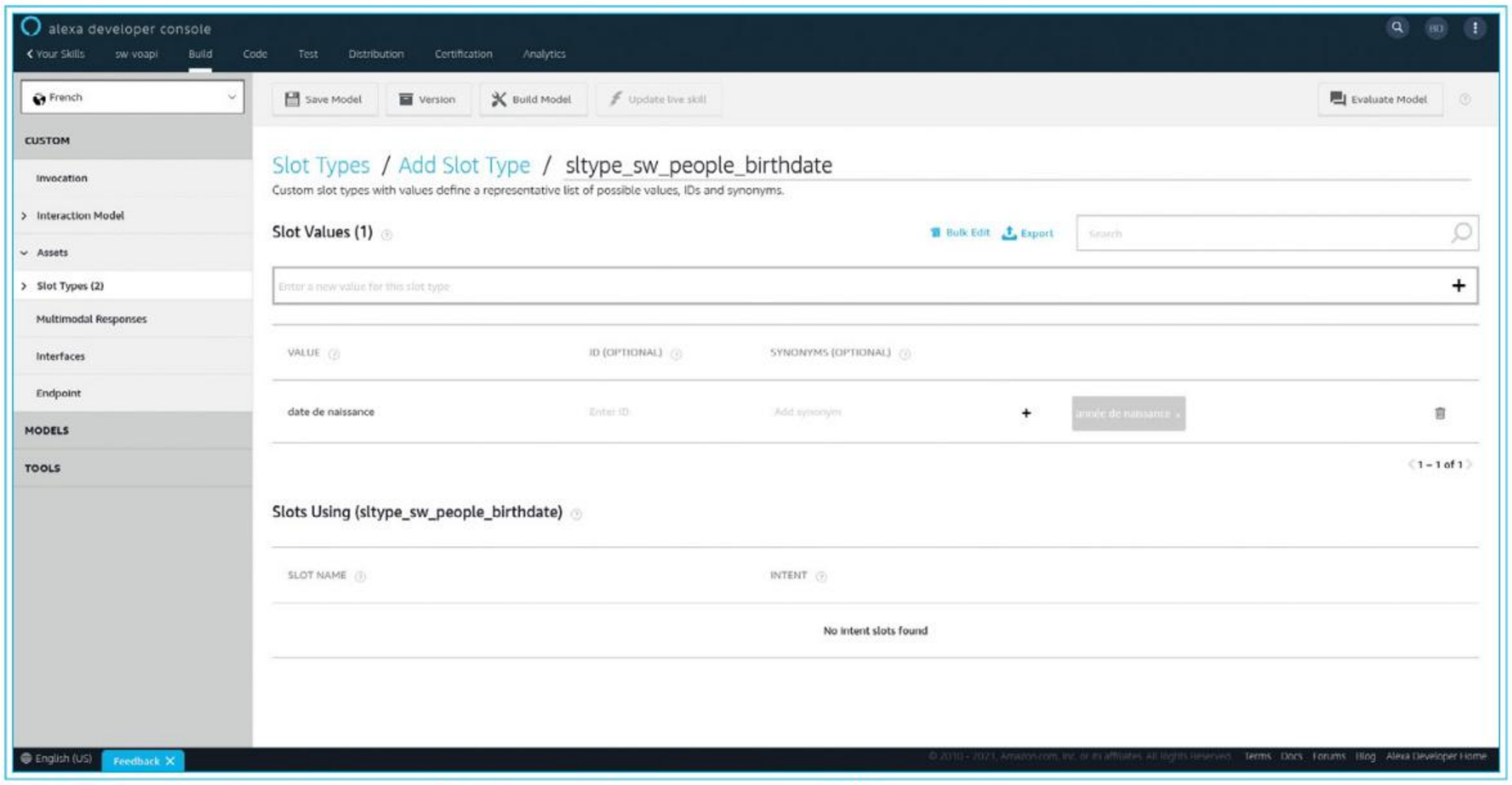


Fig. 8 : Dates de naissance toutes aussi fictives que les personnages en réalité.

- un synonyme, comme Luke pour Luke Skywalker, parce qu'il est très probable que les utilisateurs l'appellent ainsi.

Je ne vais pas m'arrêter à un seul personnage ! Je vais faire l'exercice pour les 5 plus populaires. Bien sûr, dans le cadre d'une vraie application, je devrais le faire pour tous.

Mon deuxième slot correspond à la date de naissance des personnages (figure 8).

Puisque j'aime les noms explicites à défaut d'être simples, je l'appelle **`sltype_sw_people_birthdate`**. Ensuite, je répète l'opération pour la couleur de cheveux, le fameux **`sltype_sw_people_haicolor`**.

Et enfin pour la taille, sans grande originalité : **`slype_sw_people_height`**.

2.4 Intent

D'accord, il y a un peu de vocabulaire alexéen à comprendre, mais promis, ça va devenir très vite très simple. Voici ce qu'Alexa va attendre que nous lui posions comme type de questions :

« Alexa, quelle est la couleur de cheveux de Luke sur Holo Disque ».

Ici, notre intent, c'est « quelle est la couleur de cheveux de Luke ». Or, couleur de cheveux et Luke sont deux slots, et nous pouvons reformuler la question en remplaçant ces éléments concrets par le nom des slots : « quelle est **`sltype_sw_people_haicolor`** de **`sltype_sw_people_name`** ? ». Encore une fois, tant pis si je me répète, l'IA ici n'a rien d'un tour de passe-passe. Nous définissons les différentes formulations qui permettent à un utilisateur d'interagir avec notre application ; à nous de deviner dans un premier temps quelles pourraient être ces formulations. Bien sûr, nous ne pouvons pas être exhaustifs, il y aura toujours des formulations inédites qui viendront nous surprendre. Tout comme pour les ananas, nous serons notifiés, et nous pourrons alors enrichir notre modèle.

Il est temps de créer notre intent. Va pour **`int_sw_people`** (figure 9, page suivante).

Comme je l'expliquais, je vais définir dans Sample Utterances des phrases types auxquelles notre application peut réagir. Par exemple, commençons par : « pour **`{intsl_sw_people_name}`** sa **`{intsl_sw_people_height}`** ». Notez ici l'usage des curly brackets pour utiliser un nom de slot au lieu d'un mot. C'est très simple à réaliser, il suffit d'ouvrir le curly bracket pour avoir le choix du slot que vous souhaitez utiliser (figure 10, page suivante). Appuyez sur **+**, l'intent est ajouté tel que vous l'avez défini, et les intent slots apparaissent juste en dessous (figure 11, page 30).

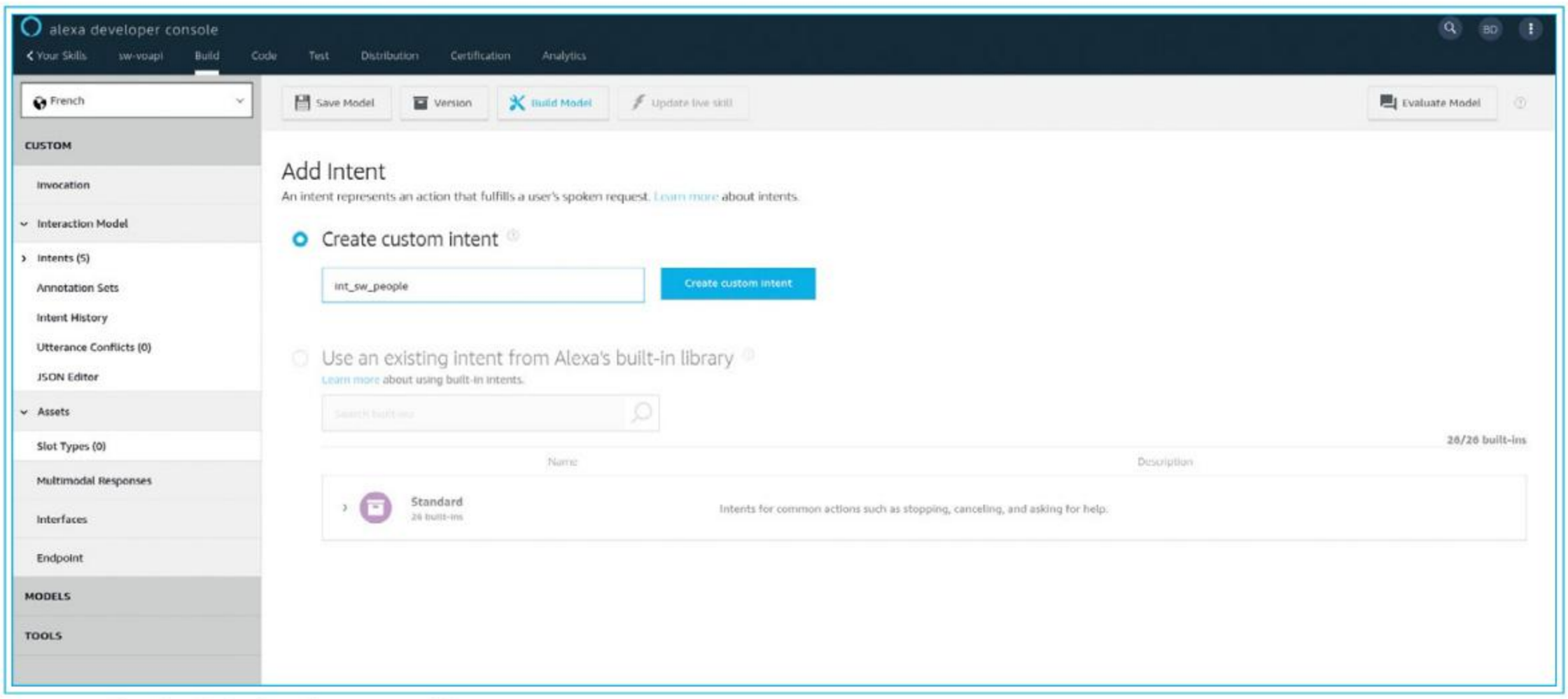


Fig. 9 : Création d'un nouvel intent.

Faisons de même pour la date de naissance : « pour {**intsl_sw_people_name**} sa **{intsl_sw_people_birthdate}** » et pour la couleur de cheveux : « pour **{intsl_sw_people_name}** sa **{intsl_sw_people_haircolor}** ».

Une fois cette tâche un peu laborieuse réalisée, il ne reste qu'à faire le lien entre l'intent slot et le slot type (figure 12, page 30).

Pour chaque intent slot, vous disposez d'une liste déroulante (figure 13, page 31) et vous n'avez plus qu'à choisir le slot type correspondant.

Quelle que soit la question qui sera posée à notre application, le nom du personnage est obligatoire ; Alexa a beau être une IA, elle ne lit pas encore dans les pensées.

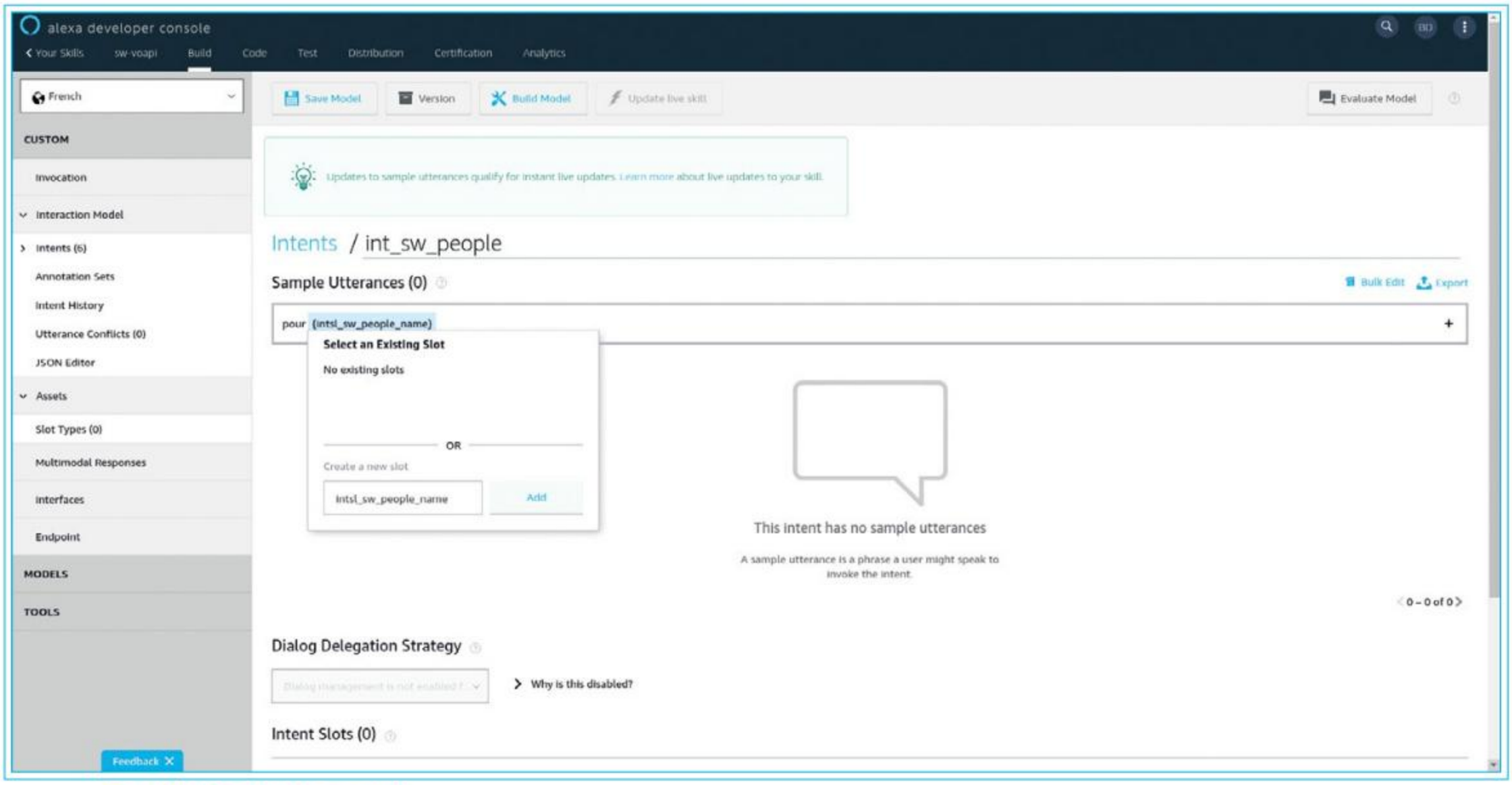


Fig. 10 : Choisissez un slot.

Chez votre marchand de journaux

et sur **www.ed-diamond.com**

PAPIER
en kiosque

FLIPBOOK HTML5
sur www.ed-diamond.com

CONNECT
LA DOCUMENTATION TECHNIQUE DES PROS DE L'IT
sur connect.ed-diamond.com

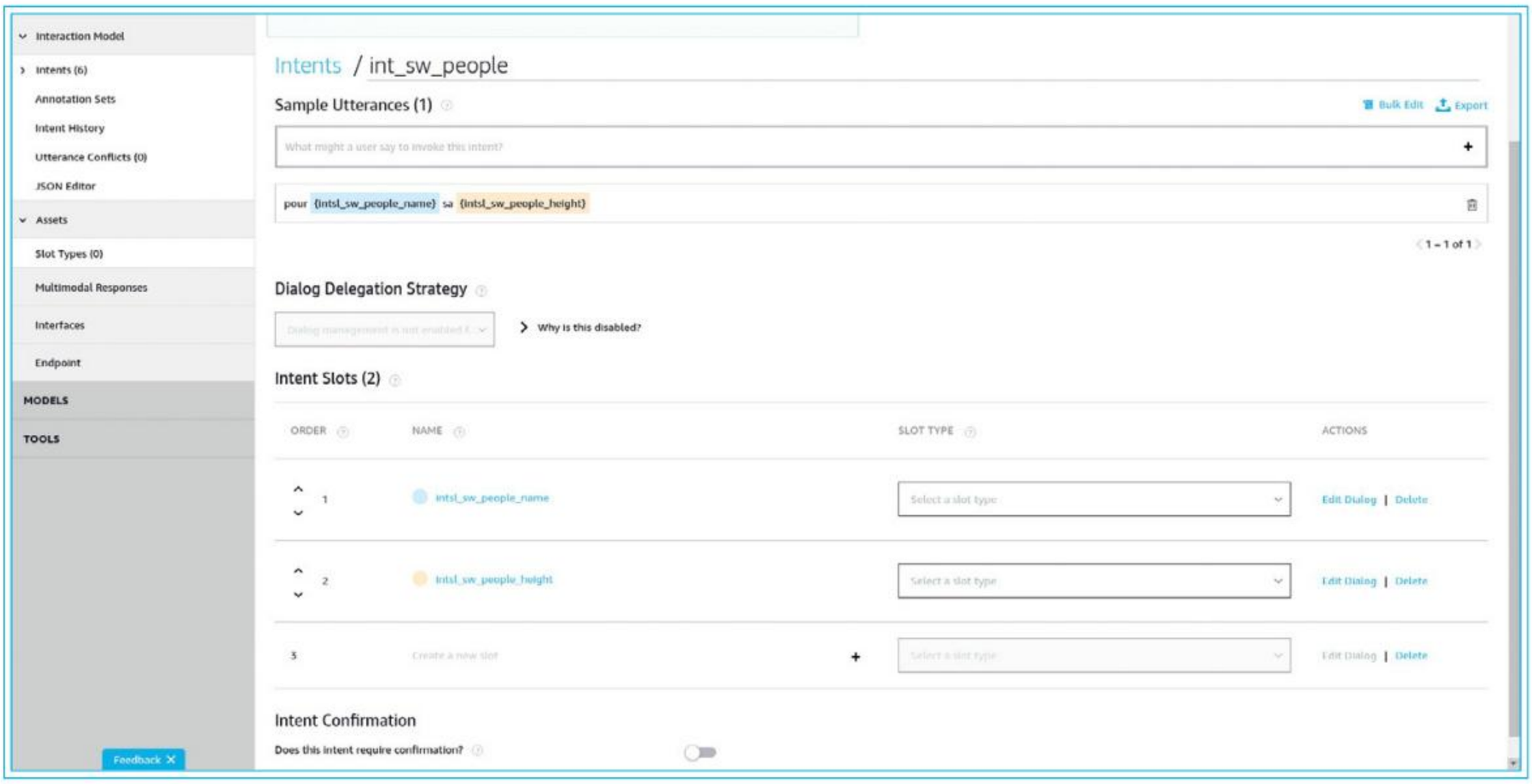


Fig. 11 : Sample Utterance.

Allez sur l'intent slot **`intsl_sw_people_name`**, et utilisez le bouton ***edit dialog*** pour le rendre obligatoire.

Pour passer à l'étape suivante, activez slot filling, et ajoutez une phrase dans Alexa Speech Prompts, puis appuyez sur **+** (figure 14, page 32).

Le champ Dialog Delegation Strategy est maintenant dégrisé, vérifiez que le mode autodélégation est bien le mode par défaut.

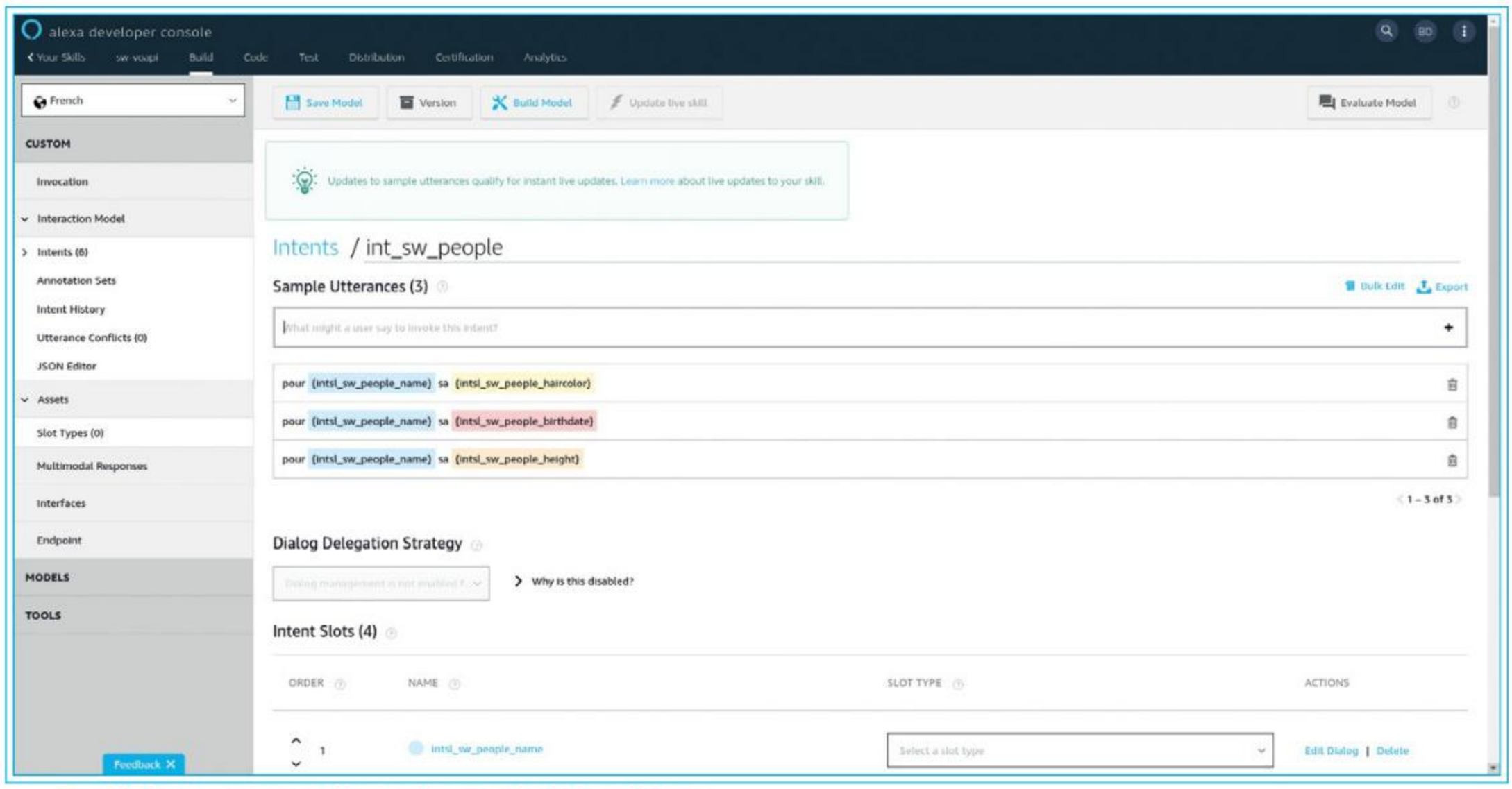


Fig. 12 : Voilà qui est fait, le tout avec de jolies couleurs.

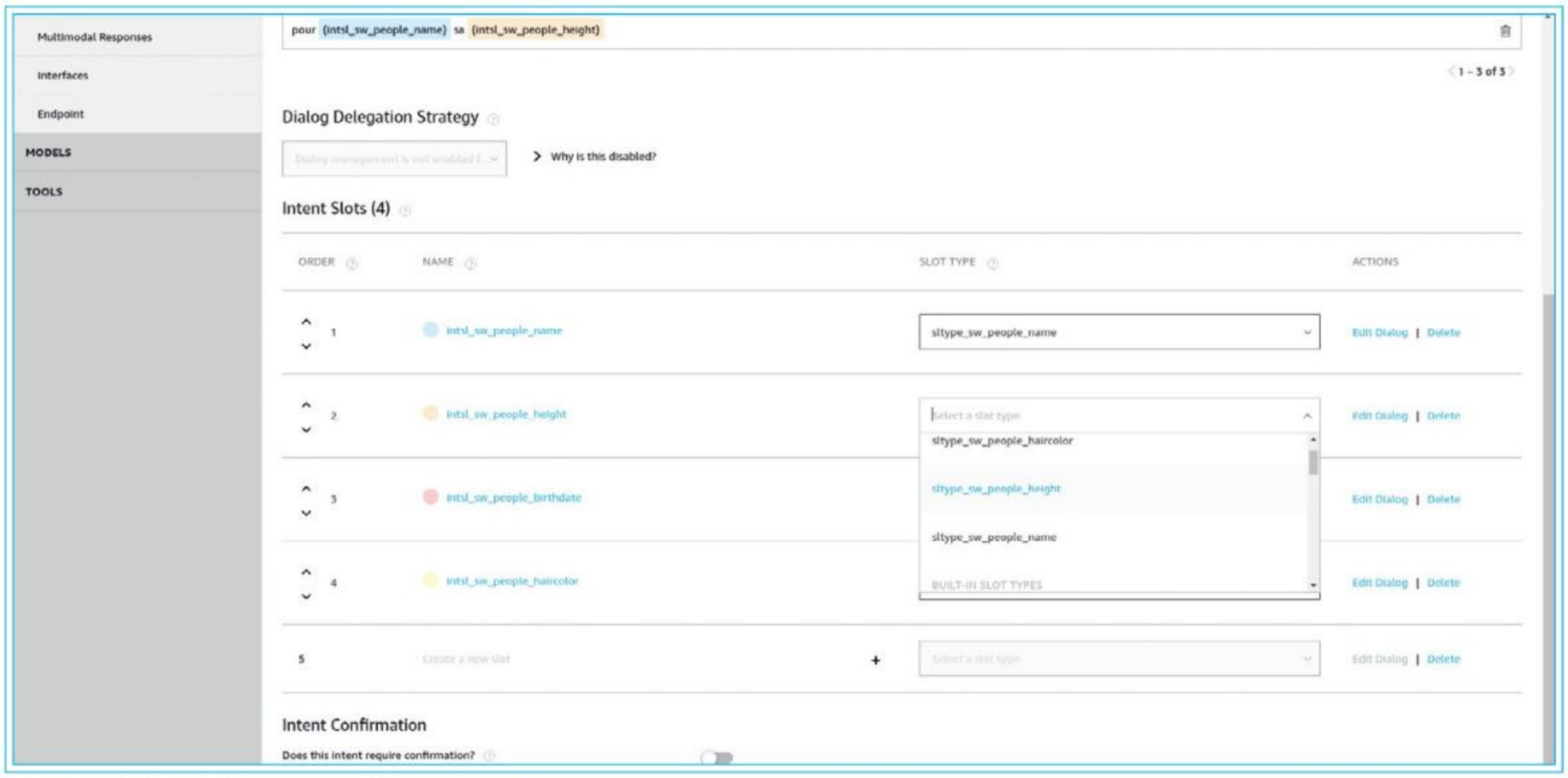


Fig. 13 : Choix du slot type par listes déroulantes.

2.5 Endpoint

Nous en avons presque fini avec Alexa Skills, mais il faut maintenant lui indiquer où elle doit envoyer les requêtes. C'est dans ***Endpoint*** que ça se définit (figure 15, page suivante).

Par défaut, le endpoint pointe sur AWS Lambda. C'est peut-être l'occasion ou jamais de s'initier aux functions as a service **[8]** qui sont particulièrement bien adaptées à ce cas d'usage : inutile d'avoir un backend qui tourne en permanence, alors qu'il suffit d'appeler une fonction quand un utilisateur vient s'amuser avec notre application. Je vous laisse tenter le coup si le cœur vous en dit, mais comme nous n'en sommes qu'aux balbutiements de notre application, pour le moment je vous propose d'utiliser un backend plus traditionnel : une URL en HTTPS.

Certains outils peuvent s'avérer particulièrement utiles pendant cette phase de développement. C'est le cas de **ngrok [9]** ; c'est un simple binaire qui vous permet de faire un tunnel entre un port de votre machine et une URL en HTTPS, ce qui tombe plutôt bien. Si vous vous imaginiez déjà devoir générer des certificats et les importer dans votre application, vous pouvez respirer. En une simple commande, j'expose le port **8080** de ma machine :

```
$ ngrok http 8080

ngrok by @inconshreveable
(Ctrl+C to quit)

Session Status                online
Account                       John Blaskowich (Plan: Free)
Version                       2.3.35
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://1d6cf4a12fd5.ngrok.io ->
http://localhost:8080
Forwarding                    https://1d6cf4a12fd5.ngrok.io
-> http://localhost:8080
```

Il ne nous reste qu'à copier l'adresse fournie par ngrok dans la partie endpoint. Ajoutez également le chemin de l'application, et comme nous en sommes à la première version, nous utilisons **/alpha/alexa**, et dernier point, sélectionnez « my development endpoint is a sub-domain... » pour la partie certificat (figure 16, page 33).

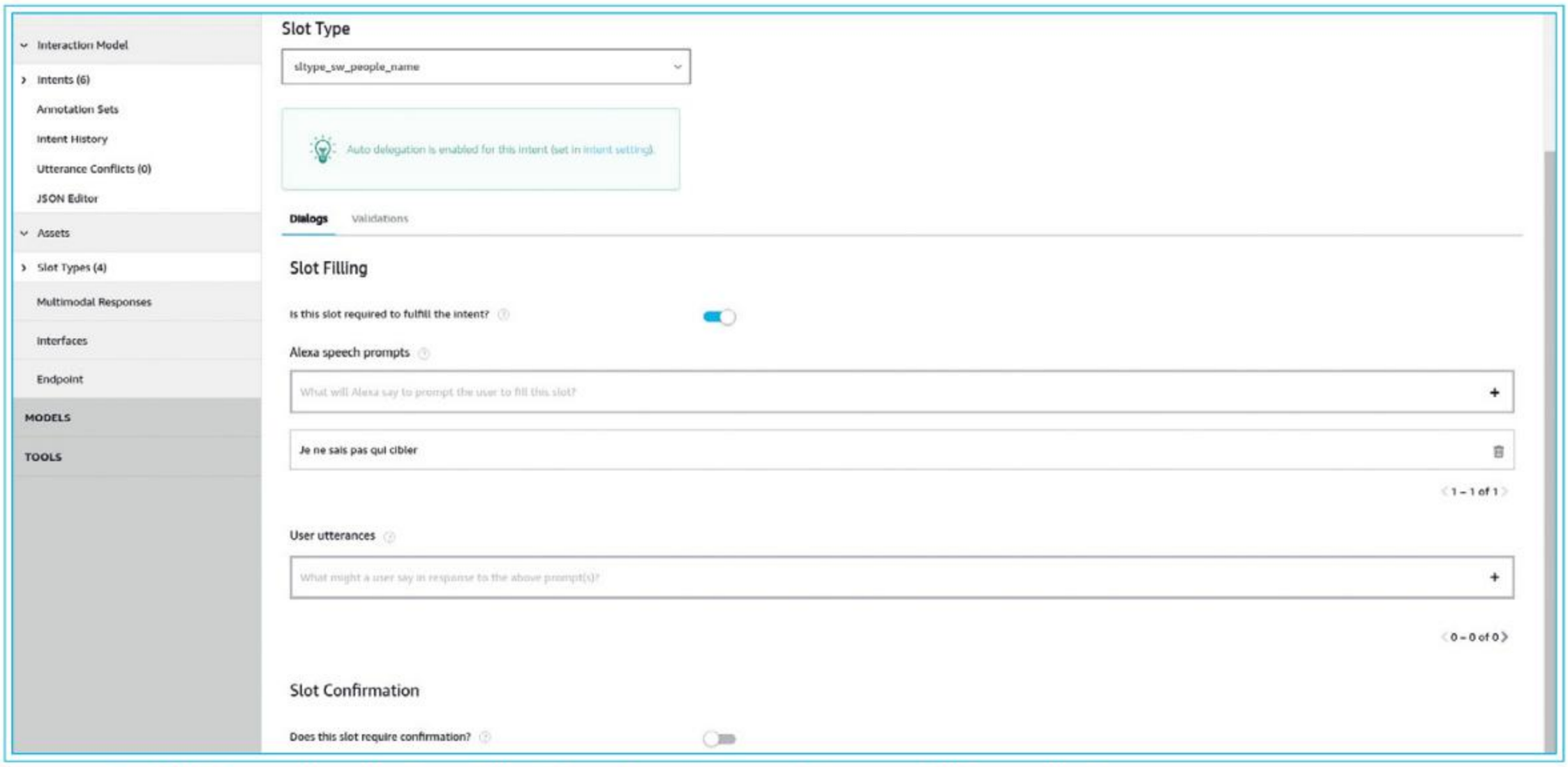


Fig. 14 : Comme dans les jeux vidéo, il y a des options secrètes pour débloquer des niveaux.

2.6 Save and Build

En revenant sur la page intent, nous pouvons maintenant sauvegarder notre travail et lancer le build du modèle, Machine Learning oblige.

Si vous ne croyez que ce que vous voyez, vous pouvez vérifier que le build est bien en cours, juste à côté du bouton ***restart build***. Notez que comme notre modèle n'est pas bien épais à ce stade, ça ne devrait pas prendre longtemps.

Enfin, quand le build s'achève, nous sommes notifiés par une popup (vraiment, ça se fait encore de nos jours ?). Nous pouvons passer à la suite l'esprit en paix (figure 17).

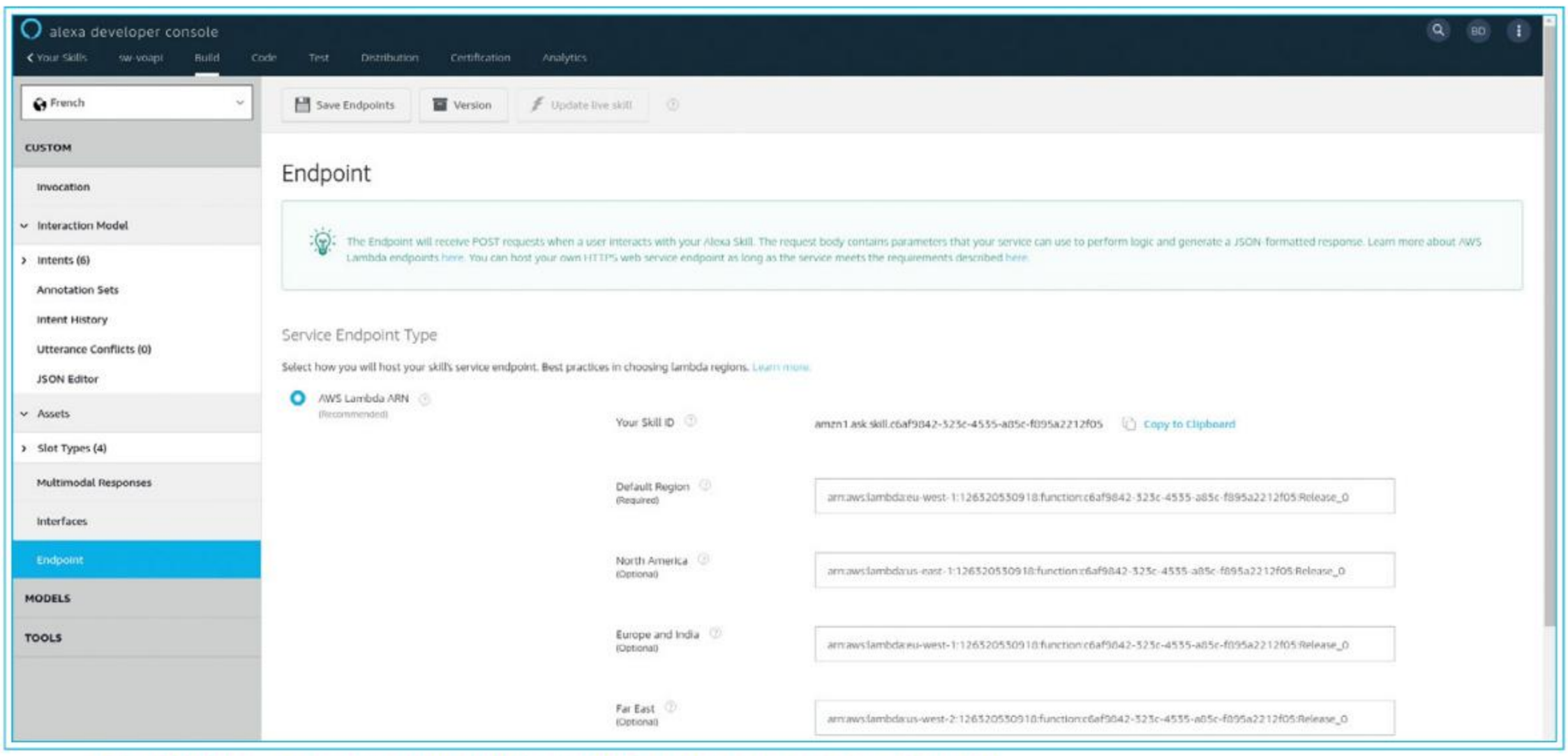


Fig. 15 : Oups, mais c'est quoi toutes ces AWS Lambda, je n'ai rien demandé.

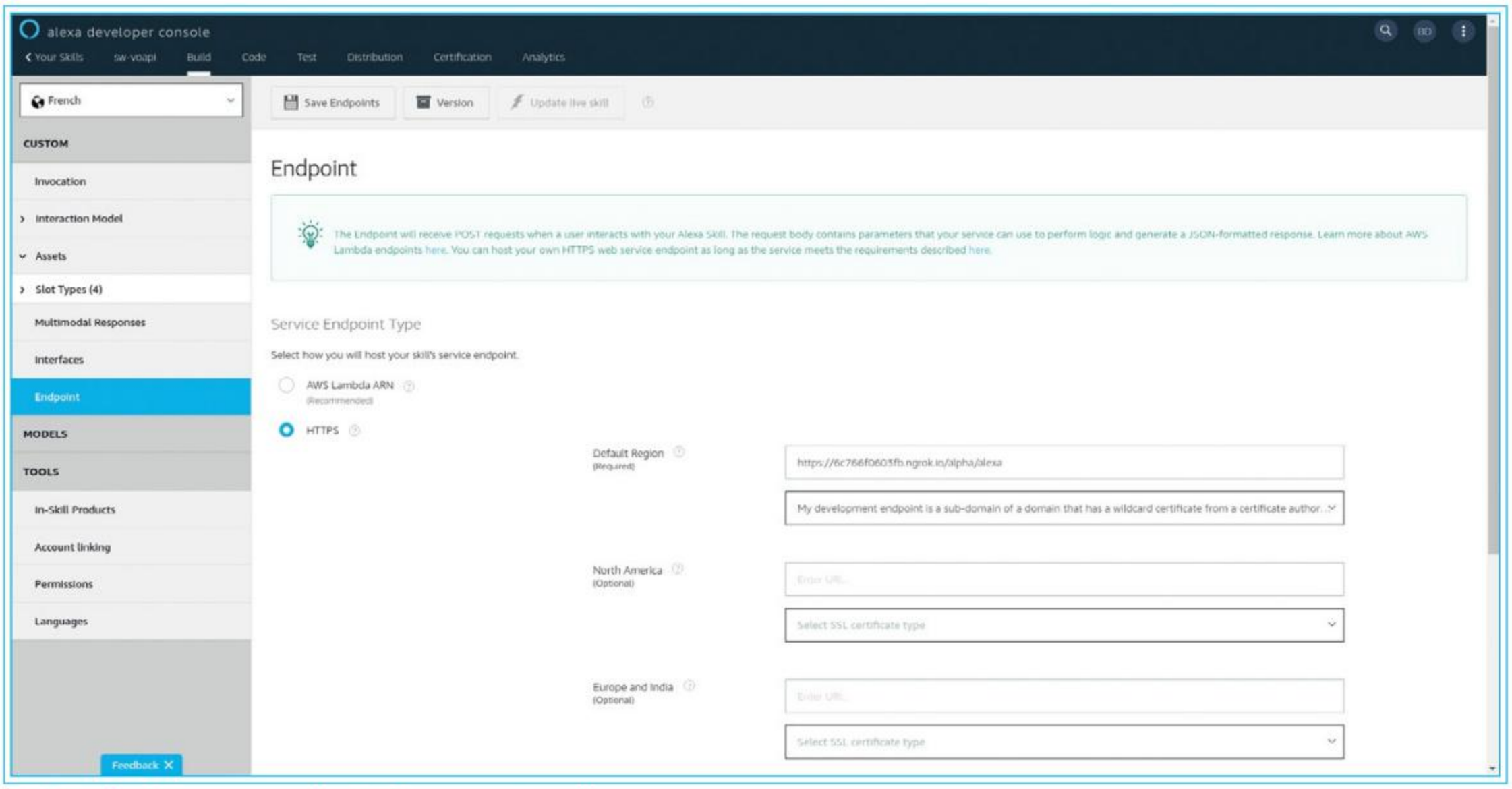


Fig. 16 : Choisissons un endpoint plus conventionnel.

3. UN PEU DE CODE

Je préfère vous mettre en garde dès à présent : je ne suis pas un développeur, ou à la limite un développeur du dimanche matin. Les rares fois où j'ai l'occasion d'écrire un peu de code, c'est soit pour un article, soit pour une démo. Alors, ne me blâmez pas pour mes erreurs, je suis le genre de personne à apprendre par l'échec, et c'est très bien ainsi.

Pour cet article, j'ai choisi d'utiliser **Golang**, histoire de tout faire à l'os, mais **Node** est sans doute l'option la plus simple pour cet exercice, même si vous pouvez tenter l'aventure avec Python ou **Java**.

Mon code est composé de trois fichiers :

- **`alexamodel.go`** qui contient les **`struct`** pour la sérialisation / désérialisation des messages de et vers Alexa Skills ;
- **`swapimodel.go`**, c'est la même chose, mais cette fois pour l'API **swapi** ;
- et le code principal et les diverses fonctions sont sans surprise dans **`main.go`**.

Fig. 17 : Des dizaines d'années plus tard, peut-être sous une forme un poil plus élégante, la célèbre popup existe encore.

Les lecteurs qui sont familiarisés avec Go n'auront sans doute pas de peine à comprendre mon code, car il est simple et stupide, et bien sûr disponible sur **GitHub [10]** ; pour les autres, je vous fais le tour du propriétaire. Il y a six fonctions :

- **`getSwapi`** : cette fonction prend en entrée un ID et une caractéristique, et va interroger swapi. J'y ai défini une variable **`peopleSearchInfo`** qui recueille les données au format **JSON** liées au personnage que nous recherchons.

  J'ai choisi d'utiliser swapi, parce que c'est une API simple, idéale pour s'exercer. Un GET sur le chemin **`/people`** me retourne le JSON que je souhaite. Nous récupérons la caractéristique que nous cherchions, et nous la retournons à une autre fonction.
- **`buildIntentResponse`** : rien d'original ici, Alexa attend une réponse au format JSON, et cette fonction est là pour lui retourner cette réponse en fonction des informations que nous avons trouvés. C'est une réponse de type **`plainText`**, la plus simple possible.
- **`alexaSlotParsing`** : c'est la fonction qui capture la question qui nous vient d'Alexa Skill : un gros JSON bien gras duquel il nous faut extraire les slots

qui sont les seules informations dont nous avons besoin ici. C'est toujours un travail de longue haleine en Go de créer un `struct` qui permet de traiter les fichiers JSON, mais ça à l'avantage d'être simple à utiliser par la suite, et extrêmement clair.

- `getSlotPeople` : sur le même principe que la fonction précédente, mais cette fois pour récupérer le nom et l'ID du personnage.

- `helloAlexa` : c'est la toute première fonction qui permet d'interagir avec Alexa. Si vous utilisez Alexa, vous lui dites « Alexa, je voudrais écouter Nick Cave and the Bad Seeds », et Alexa ira piocher dans son catalogue musical quelques titres bien agréables à écouter. Mais quand il s'agit d'une application écrite pour Alexa, ça ne marche pas exactement comme ça. Il faut « lancer » l'application avec une phrase bien spécifique ; vous vous souvenez, c'est notre fameux Holo Disque. « Alexa, lance Holo Disque », voilà ce qu'il faut dire pour lancer notre application.

  Avec cette phrase, Alexa sait que nous ne nous adressons pas directement à elle, mais à une application qui s'appelle Holo Disque. Elle passe donc la main à notre application, et une première requête va atterrir sur notre endpoint, de type « lauch request ». Il faut une fonction spécifique pour traiter cette première requête, et répondre, par exemple, « Bonjour, bienvenue sur Holo Disque. Donnez-moi le nom d'un personnage de Stars Wars et une caractéristique que vous aimeriez connaître sur lui/elle/ça ». Notez au passage avec quelle élégance Alexa manie le langage neutre !

  La prochaine interaction entre vous et Alexa devrait tout naturellement être une intent request, ce qui est le travail des autres fonctions.

- `main` : Il ne s'agit pas du fameux état américain si cher à Stephen King dont l'orthographe correcte est Maine, mais de la fonction principale de notre programme. J'y ai mis le plus simple des serveurs web : un path `/alpha/alexa`, et un port : `8080`.

4. À VOUS DE JOUER !

Malheureusement, je ne peux pas vous faire tester tout ça sur mon Alexa, et je ne suis pas sûr que mon rédacteur en chef à GNU/Linux Magazine m'autorise à fournir un cédérom avec la version papier du magazine, si seulement exemplaire papier il y a, mais rien n'est moins sûr, triste période.

Faites un clone du repo, lancez le code, utilisez ngrok comme je vous l'ai indiqué, et tout devrait fonctionner à merveille : je le sais, c'est moi-même qui l'ai testé ! Petit conseil amical, néanmoins : inutile de vous jeter sur Amazon pour commander une Alexa, ou de chercher à l'installer sur un **Raspberry** qui traîne dans un coin ; vous pouvez simplement installer l'application sur votre smartphone. C'est comme ça que j'ai fait mes premières armes, et c'est tellement plus simple.

Amusez-vous !

CONCLUSION

Nous venons de mettre en place les fondations d'une application très simple pour Alexa. Il y a une partie de l'application à réaliser dans Alexa Skills, et pour schématiser il s'agit de la partie IA, et une autre partie à réaliser en code. Cette seconde partie est aussi bête que méchante, et sert en quelque sorte de middleware entre Alexa et une API. Bien sûr, ça pourrait être l'API elle-même, mais bon, croire ou ne pas croire aux microservices, telle est la question.

Même si c'est un petit peu intimidant, vous voyez que ça n'a rien de sorcier, et qu'il est facile de se lancer sans avoir à aller lire des pages entières de documentation. Le but de cet article n'était pas de faire de vous des experts, mais de vous permettre de mettre un pied dans cette technologie qui est très prometteuse pour les années à venir. Cela va sans dire qu'il y a des applications beaucoup plus complexes qui peuvent être réalisées avec Alexa, ou il est possible d'avoir plusieurs va-et-vient entre l'application et vous pour obtenir un résultat. Pour vous donner un exemple très simple :

« Alexa, je voudrais un café. »

« Très bien, voulez-vous du sucre avec votre café ? »

« Oui, un sucre. »

« Voici votre café avec un sucre. »

Conceptuellement, ça reste un bot, et tout ce que vous avez appris avec les bots est valable avec Alexa. Sa seule particularité, c'est sa personnalité et sa joie de vivre ! ■

RÉFÉRENCES

[1] Amusante, rassurante, effrayante, je vous laisse vous faire votre propre opinion sur cette anecdote : https://nypost.com/2017/07/10/alexa-calls-cops-on-man-allegedly-beating-his-girlfriend/

[2] Nabaztag est un lapin plus charmant que crétin pour apprendre l'IoT : https://www.nabaztag.com/#

[3] Voici un assistant open source qui ne manque pas de toupet, puisqu'en plus de respecter la vie privée, il peut être installé via d'astucieux hacks sur Alexa ou Google Home : https://mycroft.ai/

[4] Autre assistant open source : Kalliope. À tester : https://github.com/kalliope-project/kalliope_starter_en

[5] Je n'en ai pas trouvé des tonnes, mais celle-ci est particulièrement intéressante ; une étude sur l'utilisabilité des assistants vocaux : https://www.nngroup.com/articles/intelligent-assistant-usability/

[6] Je suis désolé d'aimer les pubs kitch des années 90, mais celle-ci m'a toujours beaucoup amusé : « La vitesse vous manque ? » https://www.dailymotion.com/video/xt7x4

[7] Encore un autre site de fans de Star Wars : https://starwars.fandom.com/wiki/Holodisc

[8] J'ai eu plusieurs fois l'occasion d'écrire sur le FaaS, et si vous voulez un framework, peut-être pas le plus simple, mais qui a de l'avenir et qui ne soit pas hébergé par un cloud provider, je vous recommande **Knative** : https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMFHS-101/Google-Knative-le-futur-standard-du-deploiement-d-applications-serverless

[9] Je n'ai pas d'action chez l'éditeur de ngrok, cependant quand je vois un outil pratique comme celui-là, je n'hésite pas à en parler et à le suggérer : https://ngrok.com/

[10] Le code de Holo Disque sur GitHub : https://github.com/NEwa-05/voapi

POUR ALLER PLUS LOIN

Il y a un très bon livre pour débuter, mais aller bien plus loin que ne peut le faire cet article. Je vous le recommande chaleureusement, d'autant qu'il vous fait développer une application qui est portée aussi bien sur Google Home que sur Alexa. Il s'agit de « Voice User Interface Projects: Build voice-enabled applications using Dialogflow for Google Home and Alexa Skills Kit for Amazon Echo » de Henry Lee, aux éditions Packt.

INTÉGRATION CONTINUE AVEC DRONECI

THOMAS RIBOULET
[Consultant développement Backend (Ruby) et infrastructure]

MOTS-CLÉS : CI, CD, DOCKER, GIT

Après une introduction à Gitea, un outil Open Source de gestion de dépôts Git, voici une introduction à DroneCI, un outil d'intégration continue. Deuxième volet pour voir comment prendre en main cette partie clef de support au travail quotidien de toute l'équipe.

Suite à un article récemment publié sur le sujet du « self hosting » de dépôts **Git** avec la solution libre **Gitea [1]**, j'ai voulu me pencher sur une brique complémentaire : l'intégration continue. Ces dernières années, elle est devenue populaire pour les offres commerciales et certains projets libres ou Open Source de gestion de dépôts Git ont inclus une brique d'intégration continue. Pour ma part, je considère que cela rend plus complexe un outil déjà fort occupé avec les dépôts. La limitation des responsabilités est un point important pour assurer la maintenabilité. Utiliser des briques aux responsabilités limitées et bien cadrées, en tirant parti de leur bon fonctionnement de concert, est plus judicieux à de nombreux niveaux. Voyons donc pourquoi et comment installer, puis gérer un outil d'intégration continue : **DroneCI**.

1. OPEN SOURCE OU PAS ?

Sur le papier DroneCI est Open Source, mais seulement en partie. Les images **Docker** disponibles sur **Docker Hub** (et utilisées dans cet article) sont en fait des versions d'essai de la version « Entreprise ». En lisant de plus près la licence, nous pouvons voir que seule une partie du dépôt est Open Source et que la version Open Source de DroneCI est très limitée. Si la version Entreprise permet d'utiliser des grappes de runners, ce n'est pas le cas de la version Open Source, qui ne fonctionnera que sur une machine seulement.

L'essai en question court pour 32 jours calendaires, mais la licence Open Source a cependant deux limitations : si votre société fait moins de 5 millions de dollars américains de chiffre d'affaires, et si vous faites moins de 5000 builds par an.

Comme vous le verrez dans la conclusion, je pense que DroneCI est un bon produit, mais cette façon d'annoncer le produit comme Open Source, alors qu'en fait seul un core très restreint l'est, me laisse un goût amer en bouche. Au minimum, ce n'est pas ce à quoi je m'attendais (comparé à d'autres projets Open Source), mais c'est leur choix.

Donc, en lisant l'article qui suit, gardez cela en tête, et lisez bien la licence dans le dépôt **Git**. Et puis, au coin du feu, pendant les longues soirées de cette époque, posez-vous la question : « c'est quoi l'Open Source pour moi ? ».

2. POURQUOI DONC ?

Tout comme un dépôt Git, l'outil d'intégration continue est un bloc indispensable pour toute équipe de développement. La plupart des grosses solutions de hosting de dépôts Git ont ajouté une solution d'intégration continue dans leur offre, et certains fournisseurs de PaaS et IaaS font de même. De nombreuses équipes signent vite pour un « tout en un » qui promet monts et merveilles. En quelques clics, l'équipe peut commencer à travailler, c'est le rêve des startups et des managers.

Si cela peut se comprendre, ce n'est pas la seule solution. Mettre tous ses œufs dans le même panier est un risque important pour toute entreprise. Si le fournisseur en question connaît une interruption de service, votre degré d'impact sera proportionnel à votre dépendance aux services qu'il héberge. Utiliser plusieurs briques qui interagissent selon des standards ouverts permet de se donner plus de liberté sur les choix, pour chacune de ces briques.

Bien sûr, héberger et gérer soit même un dépôt Git et une chaîne d'intégration continue n'est pas sans risque non plus : vos serveurs peuvent aussi tomber en panne. Il faut aussi considérer cette question, mais utiliser des briques différentes permet de migrer progressivement afin d'établir les bonnes pratiques et procédures pour une gestion « en production » de vos outils.

Si les dépôts Git peuvent paraître sensibles et risqués de migrer vers un hébergement privé (avec Gitea par exemple), une chaîne d'intégration continue est beaucoup moins risquée. En principe, la chaîne d'intégration continue ne fait que télécharger du code source, installer des dépendances, lancer des tests et d'autres scripts avant de rendre son verdict ou d'uploader un artefact quelque part. Il n'y a rien qui ne puisse être redémarré à zéro.

Nous allons donc voir comment installer DroneCI de façon sommaire, celles et ceux qui ont leurs habitudes avec des outils comme **Puppet**, **Chef**, **Ansible**, **SaltStack**, **Terraform** & co trouveront, sans nul doute, comment installer DroneCI selon leur convenance.

3. INSTALLATION

Pour faire simple, nous allons utiliser un fichier `docker-compose.yml` et un serveur Gitea préalablement mis en place. N'ayez cependant crainte : l'installation avec GitHub, **Gitlab**, ou **BitBucket** est très similaire **[2]**.

Commencez par démarrer une machine virtuelle chez votre fournisseur préféré. Le principal est d'avoir un accès root, une adresse IP et de quoi s'assurer qu'il sera possible à votre serveur Gitea et votre serveur Drone de communiquer via des adresses IP ou des noms d'hôtes.

Créons ensuite une application OAuth via le panel d'administration de Gitea. Cela nous donnera un Client ID et un Client Secret qui vont être utilisés à l'étape suivante. L'URL de redirection doit pointer vers l'adresse IP ou le nom d'hôte du serveur Drone, agrémentée d'un `/login` (https://ci.example.com/login).

Il faut préparer un secret supplémentaire qui sera utilisé entre le serveur Drone et les instances qui vont exécuter

les tâches (runners). Pour le générer, il est possible d'utiliser la commande suivante :

```
$ openssl rand -hex 16
b2221fd8090a38720fceea26a445eca6
```

C'est une chaîne de caractères aléatoire, vous pouvez utiliser autre chose pour la générer, la seule importance est que le serveur Drone et les runners utilisent la même.

```
version: "3.3"
services:
  drone-server:
    image: drone/drone:1
    environment:
      DRONE_GITEA_CLIENT_ID: <valeur_client_
id_application_oauth_gitea>
      DRONE_GITEA_CLIENT_SECRET: <valeur_
client_secret_application_oauth_gitea>
      DRONE_GITEA_SERVER: <url_pointant_vers_
host_gitea>
      DRONE_RPC_SECRET: <secret_
supplémentaire_généré_avec_openssl>
      DRONE_SERVER_HOST: <nom_d_hôte_serveur_
drone>
      DRONE_SERVER_PROTO: https
      DRONE_USER_CREATE: username,admin:true
    ports:
      - "8080:80"
  haproxy:
      container_name: lb
      image: 'tomdess/haproxy-certbot:latest'
      environment:
          - CERTS=<nom_d_hôte_serveur_drone>
          - EMAIL=<you@example.com>
      volumes:
          - '/var/hosting/ci/letsencrypt:/
etc/letsencrypt'
          - '/var/hosting/ci/haproxy/haproxy.
cfg:/etc/haproxy/haproxy.cfg'
      ports:
          - '80:80'
          - '443:443'
      depends_on:
        - drone-server
```

Ce fichier `docker-compose.yml` permet de démarrer deux containers : un premier pour Drone lui-même, l'autre pour **HAProxy**. Les variables d'environnement à remplacer sont clairement décrites dans l'exemple, elles sont issues soit de l'application OAuth dans Gitea, soit de l'étape précédente, soit de votre setup.

Notons la présence de plusieurs variables d'environnement :

- **DRONEGITEA*** : trois variables qui viennent de Gitea et de l'application que nous avons créée ;
- **DRONE_RPC_SECRET** : le secret généré avec **OpenSSL** ;
- **DRONE_SERVER_HOST** : un nom d'hôte pour le serveur drone, il faut que ce soit un nom d'hôte public ou résolvable pour les machines concernées ;
- **DRONE_SERVER_PROTO** : le protocole à utiliser pour les requêtes faites sur le serveur Drone : HTTP ou HTTPS, ce dernier étant probablement judicieux ;
- **DRONE_USER_CREATE** : le nom d'utilisateur (sur le serveur Gitea) qui sera le premier administrateur.

Le container HAProxy est utile pour pouvoir mettre en place un certificat SSL devant l'interface web du serveur Drone. L'image du container est celle de **tomdess [3]** et permet d'utiliser **Certbot** pour avoir des certificats SSL signés à un moindre coût.

Notons les deux volumes utilisés par le container HAProxy : l'un pour le bien des certificats via Let's Encrypt et l'autre pour la configuration de HAProxy. Il faut donc veiller à lui donner un petit fichier de configuration :

```
global
    maxconn 20480
    ############# IMPORTANT ############
#####################
    ## DO NOT SET CHROOT OTHERWISE YOU
HAVE TO CHANGE THE  ##
    ## acme-http01-webroot.lua file ##
    # chroot /jail ##
    ####################################
#####################
    lua-load /etc/haproxy/acme-http01-
webroot.lua
    # SSL options
    ssl-default-bind-ciphers
AES256+EECDH:AES256+EDH:!aNULL;
    tune.ssl.default-dh-param 4096
```

```
# DNS runt-time resolution on backend
hosts
resolvers docker
    nameserver dns "127.0.0.11:53"

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms
    option forwardfor
    option http-server-close

    # never fail on address resolution
    default-server init-addr last,libc,none

frontend http
    bind *:80
    mode http
    acl url_acme_http01 path_beg /.well-
known/acme-challenge/
    http-request use-service lua.acme-
http01 if METH_GET url_acme_http01
    redirect scheme https code 301 if !{
ssl_fc }

frontend https
    bind *:443 ssl crt /etc/haproxy/certs/
no-sslv3 no-tls-tickets no-tlsv10 no-tlsv11
    http-response set-header Strict-
Transport-Security "max-age=16000000;
includeSubDomains; preload;"
    default_backend www

backend www
    server server1 172.17.0.1:8080 check
port 8080
    http-request add-header X-Forwarded-
Proto https if { ssl_fc }
```

Ainsi, le container **Drone** écoutant sur le port **8080** sera accessible à travers HAProxy sur le port **443**.

Une fois ces deux containers en route, il vous est possible d'accéder à l'interface web de Drone via https://nom-d-hote.example.com (remplacez avec le nom d'hôte ou l'adresse IP que vous avez configurés).

4. AJOUT D'INSTANCES

Mais pour que cela soit vraiment utile, il faut avoir au moins un container de runners. Il est tout à fait possible d'en lancer un sur la même machine que le serveur **Drone**, en ajoutant le service suivant dans le fichier **docker-compose.yml** :

```
  drone-runner:
    image: drone/drone-runner-docker:1
    environment:
      DRONE_RPC_SECRET: <secret_
supplémentaire_généré_avec_openssl>
      DRONE_RPC_HOST: <nom_d_hôte_serveur_
drone>
      DRONE_RPC_PROTO: https
      DRONE_RUNNER_CAPACITY: 2
      DRONE_RUNNER_NAME: bob
    ports:
      - "3000:3000"
    volumes:
      - '/var/run/docker.sock:/var/run/docker.
sock'
```

La variable d'environnement **DRONE_RPC_SECRET** doit absolument correspondre à celle définie pour le serveur **Drone**. La variable d'environnement **DRONE_RPC_HOST** doit, quant à elle, permettre d'atteindre le serveur **Drone**. La variable d'environnement **DRONE_RUNNER_CAPACITY** permet de spécifier combien de travaux ce serveur pourra exécuter en parallèle.

Une fois ce container démarré, les actions définies pourront être exécutées.

Notons l'image spécifiée : **drone/drone-runner-docker:1**. Elle permet de lancer un runner **docker**, d'autres types de runners sont utilisables : **docker**, **exec**, **ssh**, **kubernetes**... Les runners **exec** peuvent être intéressants dans certains cas qui ne sont pas adaptés à l'exécution dans un container, mais cela présente des risques, puisque les tâches ne seront plus isolées lors de leur exécution.

5. CONFIGURATION DE TRAVAUX

Drone supporte plusieurs types de pipelines : Docker, **Kubernetes** et ligne de commande (**exec**). Pour cet article, nous allons voir les pipelines Docker et ligne de commande.

Les pipelines Docker seront utilisés sur le serveur, les pipelines ligne de commande permettent d'exécuter la ou les tâches définies dans l'environnement local afin de vérifier que tout marche.

5.1 Un exemple simple

Les pipelines Drone se configurent via un fichier **YAML** dans le dépôt Git. Ce fichier **.drone.yml** est très similaire aux fichiers de configuration d'autres outils d'intégration continue. Voici un exemple très simple :

```
kind: pipeline
type: docker
name: default

steps:
- name: test
  image: debian:stable-slim
  commands:
  - echo "hello world"
```

Ce fichier définit un **pipeline** de type **docker** dont le nom est **default**. Ce pipeline contient une étape (**step**) nommée **test** qui utilise l'image docker **alpine** et qui exécutera deux commandes **echo**.

5.2 Localement

Nous pouvons tester ce pipeline sommaire localement avec l'utilitaire en ligne de commande de Drone. Disponible pour la plupart des plateformes, il s'installe rapidement **[3]**.

Une fois installé, il suffit d'exécuter cet utilitaire dans le dépôt où est présent un fichier **.drone.yml**. L'exemple précédent donnera le résultat suivant :

```
$ ./drone exec
13:06:05
[test:0] + echo "hello world"
[test:1] hello world
```

5.3 Sur le serveur

En ouvrant dans votre navigateur l'URL qui correspond à votre serveur drone (https://drone.example.com par exemple), vous aurez d'abord accès à la validation du lien entre vos instances Drone et Gitea. En suivant la redirection vers votre instance Gitea, vous pourrez autoriser l'application et une redirection vers l'interface web de Drone aura lieu. Une fois dans cette interface, il est alors possible de parcourir les dépôts Git visibles par l'utilisateur sur le serveur Gitea. Chacun d'eux est alors activable pour que l'instance de Drone puisse exécuter les tâches voulues.

5.4 Cron

Il est aussi possible de configurer des tâches programmées pour exécuter un pipeline particulier via une configuration similaire à Cron **[4]**. Elles sont configurables via l'utilitaire en ligne de commande ou via la section ***Settings*** du dépôt Git dans l'interface web de Drone.

5.5 Environnement

Comme tous les outils d'intégration continue, il y a un certain nombre de variables d'environnement qui sont accessibles dans les pipelines, certaines sont liées à Drone, mais la plupart sont liées au pipeline, au dépôt et au contexte d'exécution de la tâche elle-même. Il y a une liste de référence dans la documentation **[5]**. En particulier, nous pouvons utiliser :

- **DRONE_COMMIT** ou **DRONE_COMMIT_SHA** : l'identifiant SHA du commit courant ;
- **DRONE_COMMIT_REF** : la référence Git pour l'exécution en cours ;
- **DRONE_BRANCH** : le nom de la branche courante ;
- **DRONE_REPO** : le nom complet du dépôt Git ;
- **DRONE_REPO_LINK** : l'URL vers le dépôt Git sur l'instance Gitea (ou l'interface web de choix) ;
- **CI** : un booléen permettant à un script de vérifier que l'exécution est faite dans un environnement d'intégration continue.

6. MONTÉE EN CHARGE

Comme chaque instance d'un runner Drone ajoute une brique de capacité, il est facile de comprendre comment modifier la capacité de la flotte à volonté. Selon le choix de fournisseur d'infrastructure, il reste à décider comment et quand ajouter ou enlever de la capacité, selon les critères spécifiques.

ABONNEZ-VOUS !

11 MAGAZINES/AN

69 €*

au lieu de 97,90 €

FRAIS DE PORT OFFERTS

*Prix TTC en Euros / France Métropolitaine

DISPONIBLE EN VERSION PAPIER OU FLIPBOOK

DISPONIBLE EN VERSION PAPIER OU FLIPBOOK

Offre Papier : LM1

Offre Flipbook : LM4

69 €*

Prix kiosque : 97,90 €

Économie : 28,90 €

*Prix TTC en Euros / France Métropolitaine

DISPONIBLE EN VERSION PAPIER OU FLIPBOOK

Offre Papier : LM+1

Offre Flipbook : LM+4

129 €*

Prix kiosque : 187,30 €

Économie : 58,30 €

*Prix TTC en Euros / France Métropolitaine

DÉCOUVREZ LE FLIPBOOK ! sur : www.ed-diamond.com

FLIPBOOK HTML5

BULLETIN D'ABONNEMENT

JE M'ABONNE À
GNU/LINUX MAGAZINE

- ☐ Offre **LM1** - 11 numéros pour 69 €* (papier)
- ☐ Offre **LM4** - 11 numéros pour 69 €* (Flipbook)

JE M'ABONNE À **GNU/LINUX MAGAZINE ET SES HORS-SÉRIES**

- ☐ Offre **LM+1** - 11 numéros et 6 hors-séries pour 129 €* (papier)
- ☐ Offre **LM+4** - 11 numéros et 6 hors-séries pour 129 €* (Flipbook)

*Prix TTC en Euros / France Métropolitaine - Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

À découper ou recopier et à renvoyer avec votre réglement à :

Les Éditions Diamond Service des Abonnements
10 Place de la Cathédrale – 68000 Colmar – France

JE RÈGLE ..... €

- ☐ par **chèque bancaire** ou **postal** à l'ordre des Éditions Diamond (uniquement France et DOM TOM)
- ☐ Pour les règlements par **virement**, veuillez nous contacter **par e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20**

COORDONNÉES DE L'ABONNÉ

Société : .....
Nom : .....
Prénom : .....
Adresse : .....
.....
Code Postal : .....
Ville : .....
Pays : .....
Téléphone : .....
E-mail : .....

- ☐ J'autorise GNU/Linux Magazine à me contacter par e-mail ou par téléphone

RETROUVEZ TOUTES NOS OFFRES SUR : www.ed-diamond.com !

LISEZ DÈS À PRÉSENT TOUS NOS MAGAZINES EN LIGNE !

Simple. Rapide. Pratique.

FLIPBOOK HTML5

Tous nos magazines ont leur version Flipbook :

LINUX MAGAZINE / FRANCE · MISC · LINUX PRATIQUE · HACKABLE MAGAZINE

Rendez-vous sur www.ed-diamond.com

Notons aussi la possibilité de gérer le niveau de concurrence dans les instances de runner afin d'ajuster le nombre de tâches exécutables en parallèle par chaque runner en fonction de la capacité du serveur (virtuel ou non) qui l'héberge.

7. TOUT DOCKER

Cet article s'est centré sur un type de runner particulier : **docker**. Il utilise des containers Docker pour exécuter les tâches de type **docker** que nous définissions dans le pipeline.

Dans le cas où nous souhaiterions utiliser **DockerInDocker** pour préparer et publier des images Docker, il faudra utiliser une image **docker:dind** dans l'étape concernée.

```
kind: pipeline
type: docker
name: default

steps:
- name: test
  image: docker:dind
  commands:
  - docker build -t
```

8. EXEMPLES CHOISIS

Voyons maintenant comment configurer des pipelines types afin de voir comment utiliser DroneCI pour des cas simples, mais qui sont à la base de la plupart des usages.

Nous allons utiliser le cas d'une application **Ruby**, mais les concepts décrits sont réutilisables dans la plupart des langages.

8.1 Un pipeline simple avec du cache

Une application Ruby nécessite souvent de nombreuses gems et leur installation avec **bundler** peut prendre un certain temps. Il est donc utile d'avoir un moyen d'utiliser un cache pour permettre de réutiliser d'une étape à une autre le travail fait.

Pour cela, il faut utiliser un volume Docker pour stocker les gems installées dans une étape et les réutiliser automatiquement à la suivante :

```
kind: pipeline
name: default

steps:
- name: install
  image: ruby
  volumes:
  - name: bundle
    path: /usr/local/bundle
  commands:
  - bundle install --jobs=3 --retry=3

- name: test
  image: ruby
  volumes:
  - name: bundle
    path: /usr/local/bundle
  commands:
  - bundle exec rspec spec

volumes:
- name: bundle
  temp: {}
```

Nous avons donc deux étapes : l'une s'occupe de l'installation des dépendances, l'autre s'occupe de lancer les tests. Notons la définition du volume en bas du fichier et son usage dans les deux étapes dans des sections **volumes** spécifiques.

8.2 Plusieurs versions et pipelines

Et si nous souhaitons tester plusieurs versions de Ruby, nous pouvons utiliser la fonctionnalité « multipipeline » de DroneCI pour cela :

```
---
kind: pipeline
name: ruby-3

steps:
- name: test
  image: ruby:3
  commands:
  - bundle install --jobs=3 --retry=3
  - bundle exec rspec

---
```

```
kind: pipeline
name: ruby-2-7

steps:
- name: test
  image: ruby:2.7
  commands:
  - bundle install --jobs=3 --retry=3
  - bundle exec rspec
```

Ici, nous n'avons pas utilisé de cache simplement pour rester concis, rien ne nous empêche de définir un cache pour chacun des pipelines.

Il y a aussi une possibilité d'utiliser **Jsonnet [6]** afin de rendre la syntaxe moins verbeuse.

8.3 Environnement et secrets

Il est possible de définir des variables d'environnement pour chacune des étapes du pipeline :

```
steps:
- name: backend
  image: ruby:3
  environment:
    ARCH: linux
    RACK_ENV: test
  commands:
  - bundle install
  - bundle exec rspec spec
```

Mais si ces variables d'environnement ne doivent pas être stockées dans le gestionnaire de versions, il est possible de définir dans l'interface web de DroneCI, par dépôt, des secrets qui seront alors accessibles dans les pipelines.

```
kind: pipeline
name: default

steps:
- name: build
  image: ruby:3.0
  environment:
    IRC_BOT_TOKEN:
      from_secret: irc_bot_token
```

Pour les définir, il suffit d'aller dans la liste des dépôts dans l'interface web de DroneCI (la page d'accueil), puis dans les ***Settings*** et la section ***Secrets***. Notons que si le nom de la variable est défini en majuscules dans l'interface web, il faut aussi utiliser des majuscules dans l'appel de celle-ci dans le fichier YAML.

Il est aussi possible de définir des secrets au niveau de l'organisation via l'utilitaire en ligne de commande de **`drone`** **[7]**.

Notons que, par défaut, les secrets ne sont pas exposés aux Pull Requests et autres Merge Requests afin de les protéger, si le dépôt est public. Il faut cocher une petite case afin d'autoriser l'exposition de ces secrets dans le cas des PR et MR.

8.4 Conditions

Un outil d'intégration continue ne serait pas complet s'il n'a pas une fonctionnalité permettant de limiter les cas dans lesquels une étape du pipeline est exécutée. DroneCI permet de définir des conditions basées sur le nom de la branche, de la référence Git, de l'événement Git, de l'instance (la valeur du hostname de la machine hébergeant le runner), le statut, la cible et quelques autres cas **[8]**.

```
kind: pipeline
type: docker
name: default

steps:
- name: build
  image: ruby:3.0
  commands:
  - bundle install
  - bundle exec rspec
  when:
    branch:
    - master
    - feature/*
```

Cette étape de build ne sera exécutée que pour la branch **`trunk`** et les branches commençant par **`feature/`**.

Dans le cas d'événements, il est possible d'exécuter une étape en fonction de l'événement envoyé par le dépôt Git : **`push`**, **`pull_request`**, **`tag`**, **`promote`**, **`rollback`** :

```
when:
  event:
  - push
  - pull_request
```

Il est aussi possible d'utiliser les modificateurs **include** et **exclude** afin de limiter rapidement les conditions sans en lister de trop. Cela est utilisable pour quasiment tous les types de filtres (branche, référence, événement, dépôt...).

Voici un exemple incluant tous les dépôts dont le nom commence par **fix/** :

```
when:
  repo:
    include:
    - fix/*
```

Ou, pour des événements, ici en excluant les **pull_request** :

```
when:
  event:
    exclude:
    - pull_request
```

8.5 Services

Autre nécessité d'un outil d'intégration continue : les services. Ce sont les services nécessaires à l'exécution des étapes du pipeline, tels qu'une base de données.

Il faut utiliser une section **services** dans le fichier de configuration du pipeline. Ici, nous utilisons toujours des containers Docker, il faut donc utiliser une image Docker accessible par le pipeline.

```
kind: pipeline
type: docker
name: default

services:
- name: db
  image: postgresql
```

Notons que, depuis chaque étape du pipeline, le service en question sera accessible via le nom d'hôte **db**.

Une autre façon de définir un service est d'utiliser le mode détaché pour une étape. Celle-ci permet de démarrer un container en tâche de fond, par rapport aux autres containers du pipeline.

```
kind: pipeline
name: default

steps:
- name: kvstore
  image: redis
  detach: true

- name: ping
  image: redis
  commands:
  - redis-cli -h kvstore ping
```

N'oublions pas que ces services peuvent prendre du temps pour démarrer, il faut donc parfois inclure un peu de temps d'attente dans les étapes qui s'en servent.

8.6 Dépendances

Certaines étapes peuvent dépendre d'autres étapes dans un pipeline. Il est possible de définir un tel lien via une section **depends_on** dans le fichier YAML. En reprenant ici une partie du premier exemple, nous nous assurons que l'étape **test** sera exécutée après l'étape **install**.

```
steps:
- name: install
  image: ruby
  volumes:
  - name: bundle
    path: /usr/local/bundle
  commands:
  - bundle install --jobs=3 --retry=3

- name: test
  image: ruby
  volumes:
  - name: bundle
    path: /usr/local/bundle
  commands:
  - bundle exec rspec spec
  depends_on:
  - install
```

9. EXTENSIONS

Drone ayant un code publié et une possibilité directe de proposer des patchs, il y a beaucoup de choses qui sont faites par la communauté. Il y a notamment un nombre important d'extensions disponibles pour Drone pour l'intégrer avec de nombreux autres outils tels que **Slack**, **IRC**, **Vault**... La documentation de DroneCI **[2]** liste une partie de ceux-ci et renvoie vers le registre qui permet de les découvrir tous.

CONCLUSION

DroneCI est, finalement, relativement simple à mettre en place et à gérer pour la plupart des projets. Comme tout repose sur des containers Docker, il y a finalement peu de limitations à ce qui est faisable. Le fait que toutes les tâches sont définies dans des fichiers YAML contenus dans les dépôts Git des projets permet de gérer celles-ci sans avoir à se connecter et à configurer un serveur particulier. Chaque projet, chaque équipe, peut donc gérer la chaîne d'intégration continue, sans dépendre d'une autre entité.

Nous avons couvert une partie seulement des capacités de DroneCI, il y a notamment bien plus à découvrir au sujet de la syntaxe du fichier de configuration YAML. Nous devrions quand même avoir vu les cas les plus utilisés, ceux qui nous permettent de mettre en place un ou plusieurs pipelines classiques.

Cet article a montré la relative facilité d'installation, de configuration et d'utilisation d'une instance de DroneCI. La documentation n'est pas disponible en français, mais elle est assez claire sans être succincte. Elle couvre l'installation, la configuration, l'usage et l'extension de DroneCI. Elle contient aussi une référence de la syntaxe YAML utilisable pour les différents pipelines, l'API et l'utilitaire en ligne de commande.

Entre la simplicité d'installation, d'utilisation et l'extensibilité, DroneCI est une alternative plus que viable et fiable par rapport aux offres commerciales disponibles. Son usage en duo avec Gitea peut permettre à la plupart des équipes d'avoir un outil de gestion de code source, et une chaîne d'intégration et de déploiement continue sans casser la tirelire ni dépendre d'un tiers, qui sera potentiellement acheté dans 6 mois par un géant de l'industrie.

De plus, le fait que ces deux projets soient centrés sur une seule chose à la fois les maintient, de fait, dans une simplicité bienvenue. Si demain nous souhaitions remplacer l'un des deux, cela serait tout à fait faisable, cela nous laisse donc aussi une plus grande liberté.

Revenons-en à la licence, elle représente une politique Open Source quelque peu différente de ce à quoi je m'attendais. En voyant « Open Source » sur le site web, je m'étais mis en tête que DroneCI, dans sa version accessible publiquement sur GitHub ou DockerHub, était une version complète avec la très grande majorité des fonctionnalités. Je pensais que l'équipe derrière ce produit se rémunérait principalement sur les abonnements mensuels de la version Cloud (gérée par eux) ainsi que les versions « On Premise » qu'ils mettent en place pour et avec leurs clients, comme c'est le cas pour d'autres produits. C'est donc une bonne piqûre de rappel sur le sujet des politiques Open Source, sujet très d'actualité après les changements de licence d'**ElasticSearch** et de quelques autres produits aussi. ■

RÉFÉRENCES

[1] T. RIBOULET, « Hébergement privé de dépôts Git », GNU/Linux Magazine n°109, juillet 2020 : https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMFHS-109/Hebergement-prive-de-depots-Git

[2] Documentation d'installation du serveur Drone pour différents fournisseurs de gestionnaire de versions : https://docs.drone.io/server/overview/

[3] Container docker HAProxy avec Let's Encrypt : https://github.com/tomdess/docker-haproxy-certbot

[4] Installation de l'utilitaire de ligne de commande de DroneCI : https://docs.drone.io/cli/install/

[5] Documentation des tâches Cron : https://docs.drone.io/cron/

[6] Jsonnet, scripts dans les fichiers de configuration DroneCI : https://docs.drone.io/pipeline/scripting/jsonnet/

[7] Utilitaire en ligne de commande pour DroneCI : https://github.com/drone/drone-cli

[8] Conditions dans le fichier de configuration d'un pipeline DroneCI : https://docs.drone.io/pipeline/docker/syntax/conditions/

Chez votre marchand de journaux

et sur **www.ed-diamond.com**

PAPIER
en kiosque

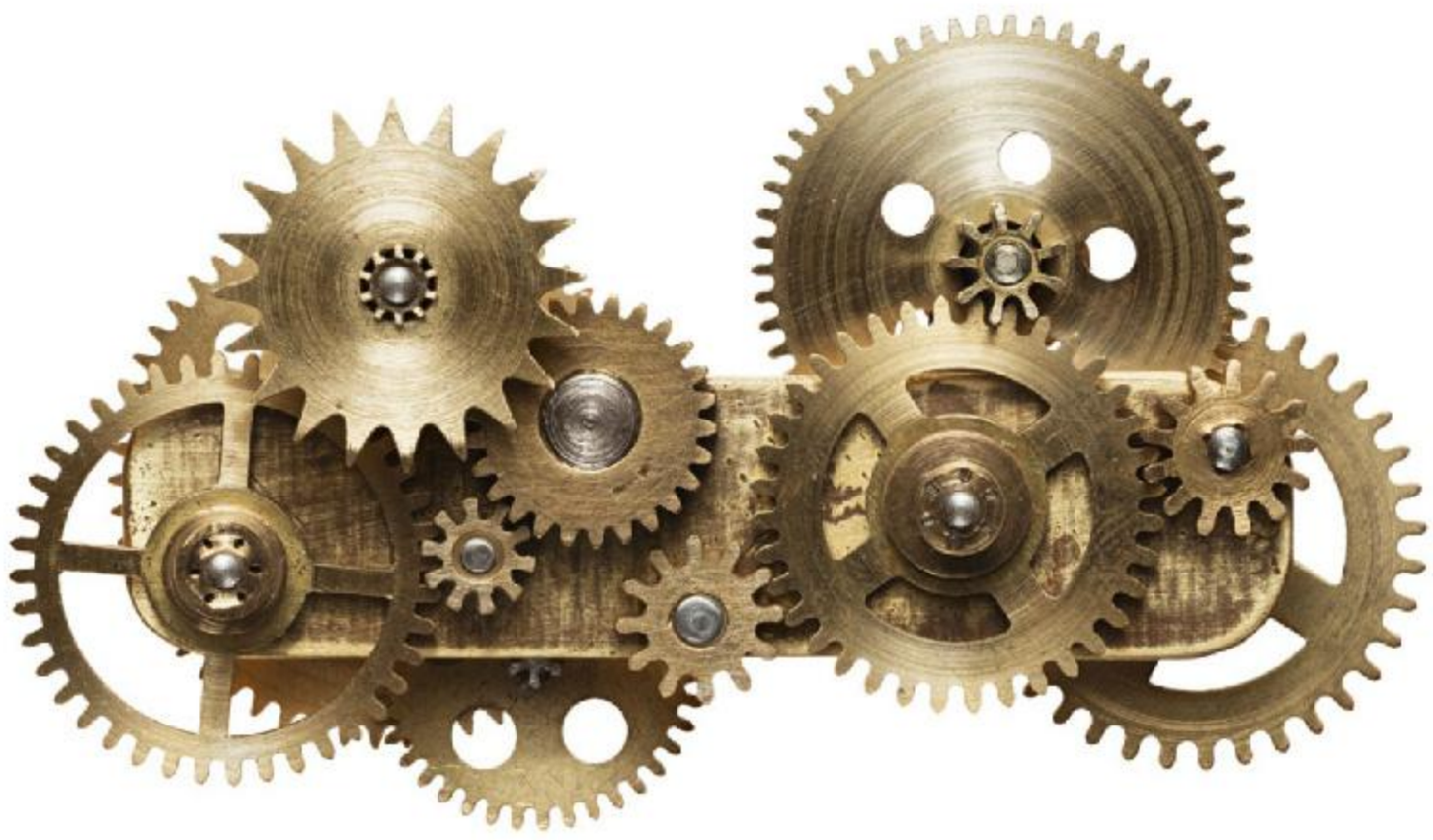
FLIPBOOK HTML5
sur www.ed-diamond.com

CONNECT
LA DOCUMENTATION TECHNIQUE DES PROS DE L'IT
sur connect.ed-diamond.com

LES MÉCANISMES « EXOTIQUES » DE PYTHON

TRISTAN COLOMBO

MOTS-CLÉS : PYTHON, SYNTAXE, POO



Certains mécanismes de Python sont méconnus, peu utilisés. Ce n'est pas pour autant qu'ils sont inutiles, loin de là ! Dans cet article, nous allons faire un tour de quelques-unes de ces structures sous-employées.

Lorsque l'on utilise un langage de programmation, on tombe dans les mêmes travers que lorsque l'on écrit ou lorsque l'on parle : on emploie souvent les mêmes tournures de phrases, les mêmes expressions. Chez quelqu'un de cultivé, lisant beaucoup, cela se remarquera moins puisqu'il a à sa disposition un vocabulaire plus riche et des tournures grammaticales plus complexes, mais pour autant, nous avons tous ce défaut.

En programmation, c'est exactement la même chose. Plus on avance dans la connaissance d'un langage, plus on connaît sa syntaxe et plus on peut varier le code que l'on écrit. Ici, en plus d'une simple notion de style littéraire, on aura bien souvent une optimisation du temps de développement, d'exécution ou des deux. De plus, il faut bien l'ajouter, pour certains d'entre nous, un code peut être « beau », bien écrit, procurer du plaisir à sa lecture (on pourrait presque le rapprocher de la poésie). Certes, on tombe plus souvent sur des brouillons de cancres que sur des poèmes, mais ça existe.

Tout cela m'amène à vous présenter dans cet article des éléments syntaxiques peu usités de **Python**. J'ai qualifié ces éléments d'« exotiques », ne sachant trop quel nom leur donner. Il est probable que le développeur Python expérimenté en connaisse et en utilise déjà quelques-uns. Cet article ne va pas révolutionner votre façon d'écrire du code Python, mais, je l'espère, enrichir vos connaissances et vous permettre de trouver dans certaines situations des solutions élégantes. La version de Python utilisée dans cet article est Python 3.9.

1. LE POLYMORPHISME

Parler de polymorphisme pour un langage à typage dynamique peut paraître particulièrement étrange. Pourtant, c'est tout à fait possible ! On peut surcharger une fonction (overload en anglais) en définissant plusieurs fois cette fonction avec des signatures différentes. Ce mécanisme est surtout connu et utilisé en Programmation Orientée Objet, mais est disponible dans le paradigme impératif de Python.

Nous allons voir que c'est le module standard **functools** qui va nous permettre de réaliser ces opérations.

1.1 Surcharge d'une fonction

Pour surcharger une fonction, on commence par définir une fonction générique pour laquelle on ne spécifie pas les types des paramètres. Cette fonction sera décorée par le décorateur **@singledispatch**.

Ensuite, on définit autant de fonctions que souhaité en spécifiant les types des différents paramètres. Ces fonctions se nommeront _ et seront décorées par **@<nom_fonction_originelle>.register**.

Prenons pour exemple une fonction **display()** affichant des messages différents en fonction du type des deux paramètres **param_1** et **param_2** qui lui sont transmis (fichier **overload.py**) :

```
from functools import singledispatch


@singledispatch
def display(param_1, param_2) -> None:
   print('Fonction générique')
   print('Paramètres :')
   print(f'- param_1 {param_1}')
   print(f'- param_2 {param_2}')


@display.register
def _(param_1: int, param_2: int) -> None:
   print('Fonction avec entiers')
   print(f'{param_1} + {param_2} = {param_1 +
param_2}')


@display.register
def _(param_1: list, param_2: int) -> None:
   print('Fonction avec une liste et un entier')
   print(f'Entier : {param_2}')
   print('Éléments de la liste :')
   for elt in param_1:
        print(elt)


if __name__ == '__main__':
    display('hello', 10)
    display(2, 3)
    display([1, 2, 3, 4], 4)
```

À l'exécution, ce sont bien les différentes définitions de **display()** qui sont exécutées en fonction du type des paramètres transmis :

```
$ python3 overload.py
Fonction générique
Paramètres :
- param_1 hello
- param_2 10
Fonction avec entiers
2 + 3 = 5
Fonction avec une liste et un entier
Entier : 4
Éléments de la liste :
1
2
3
4
```

Attention toutefois, car si le mécanisme paraît tout à fait satisfaisant, il souffre tout de même d'une contrainte majeure : il est impossible de définir plusieurs signatures dont les premiers paramètres ont le même type. Reprenons l'exemple précédent : nous avons défini une fonction **display(<int>, <int>)**, il sera donc impossible de définir une nouvelle fonction **display(<int>, <autre_type>)**. En voici la preuve :

```
...

@display.register
def _(param_1: int, param_2: int) -> None:
    print('Fonction avec entiers')
    print(f'{param_1} + {param_2} = {param_1 +
param_2}')


@display.register
def _(param_1: int, param_2: str) -> None:
    print('Fonction avec un entier et une string')
    print('Paramètres :')
    print(f'- param_1 {param_1}')
    print(f'- param_2 {param_2}')
...

if __name__ == '__main__':
    display('hello', 10)
    display(2, 3)
    display([1, 2, 3, 4], 4)
```

L'appel à **display(2, 3)** ne lancera pas **display(<int>, <int>)** comme attendu, mais **display(<int>, <str>)**.

1.2 Surcharge d'une méthode

Le même mécanisme de surcharge est disponible pour les objets. On utilise cette fois-ci le décorateur `@singledispatchmethod`, mais le principe reste le même :

```
from functools import singledispatchmethod


class Overload:
    def __init__(self, value):
        self._value = value


    @singledispatchmethod
    def display(self, param) -> None:
        print('Méthode générique')


    @display.register
    def _(self, param : int) -> None:
        print('Avec un int')


    @display.register
    def _(self, param : str) -> None:
        print('Avec une string')


if __name__ == '__main__':
    d = Overload(5)
    d.display([1, 2, 3])
    d.display(4)
    d.display('hello')
```

Le résultat est bien celui attendu :

```
$ python overload.py
Méthode générique
Avec un int
Avec une string
```

Il est intéressant de noter que les signatures des différentes fonctions ne doivent pas nécessairement comporter toujours le même nombre de paramètres :

```
...
    @display.register
    def _(self, param : bool, param_2 : int) -> None:
        print('Avec un boolean et un int')
...
```

Toutefois, la même contrainte que précédemment existe, en ce qui concerne la définition de plusieurs signatures commençant par un paramètre ayant le même type.

2. DE LA POO DANS DE L'IMPÉRATIF : AJOUTER DES ATTRIBUTS À UNE FONCTION

Puisque nous venons d'utiliser la POO, restons-y tout en passant en programmation impérative. Mais comment est-ce possible, vous demandez-vous ? Il ne faut pas oublier qu'en Python, tout est objet, donc en fait, même lorsque vous programmez en Python en utilisant le paradigme impératif, il y a de la POO sous-jacente.

Une fonction est un objet comme un autre. Vous pouvez vous en rendre compte en lançant un Shell Python :

```
>>> def fct():
...     print('Fonction')
...
>>> type(fct)
<class 'function'>
>>> dir(fct)
['__annotations__', '__call__', '__class__',
'__closure__', '__code__', '__defaults__', '__
delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__get__',
'__getattribute__', '__globals__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__
kwdefaults__', '__le__', '__lt__', '__module__',
'__name__', '__ne__', '__new__', '__qualname__',
'__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__
subclasshook__']
```

Donc, si une fonction est un objet, pourquoi ne pas lui ajouter des attributs ? C'est ce que nous allons faire dans l'exemple suivant :

```
from datetime import datetime


def add(x : int, y : int) -> int:
    add.param_x = x
   add.param_y = y
    add.timestamp = datetime.now()

    return x + y

if __name__ == '__main__':
    print(f'Addition :{add(2,3)}')
    print(f'Les paramètres de add() étaient
{add.param_x} et {add.param_y}')
    print(f'Appel de add() le {add.timestamp}')
```

On peut ainsi transmettre des données, en plus du résultat naturel de la fonction retourné par `return` :

```
$ python3 function_attr.py
Addition :5
Les paramètres de add() étaient 2 et 3
Appel de add() le 2021-04-14 10:31:33.139055
```

3. ELLIPSE

L'ellipse, ou ellipsis en anglais, se note `...`. En Python, on peut l'utiliser de différentes manières.

3.1 En remplacement de pass

Lorsqu'une fonction ou une méthode doit être écrite, on peut commencer par coder sa signature, puis laisser le bloc principal vide en utilisant la commande **pass** :

```
def fct_incomplete(arg_1 : int, arg_2 :
str) -> int :
    pass
```

Dans ce cas, il est possible de remplacer **pass** par une ellipse, le comportement du code sera identique :

```
def fct_incomplete(arg_1 : int, arg_2 :
str) -> int :
    ...
```

3.2 En tant que valeur par défaut

Il n'est pas rare d'employer des valeurs par défaut dans les paramètres d'une fonction. En général, on utilise **None** pour indiquer un paramètre non défini. Mais imaginons que **None** puisse être une valeur attendue. Dans ce cas, on peut bien sûr choisir une valeur arbitraire et indiquer dans les commentaires que, par exemple, **-32767** est la valeur indiquant une absence de valeur... Sinon il y a l'ellipse, qui peut d'ailleurs être testée :

```
def fct(param = ...) -> None:
    if param is ...:
        print('Pas de paramètre')
    elif param is None:
        print('None')
    else:
        print(f'Param : {param}')


if __name__ == '__main__':
    fct()
    fct(None)
    fct(12)
    fct(...)
```

Le résultat obtenu est :

```
$ python3 ellipsis.py
Pas de paramètre
None
Param : 12
Pas de paramètre
```

4. STRUCTURE FOR/ELSE

Ce dernier mécanisme est sans doute le plus connu des quatre présentés dans cet article. Il est possible d'exécuter une boucle **for** et, si celle-ci se déroule sans interruption (sans appel à **break**), alors le bloc **else** sera exécuté :

```
def boucle(n : int) -> None:
    for i in range(n):
        print(i)
        if i == 10:
            print('STOP')
            break
    else:
        print('Boucle terminée intégralement')
```

Avec **boucle(5)**, la boucle se déroule intégralement :

```
0
1
2
3
4
Boucle terminée intégralement
```

Par contre avec **boucle(11)**, la boucle est interrompue et le code du bloc **else** n'est pas exécuté :

```
0
1
...
8
9
10
STOP
```

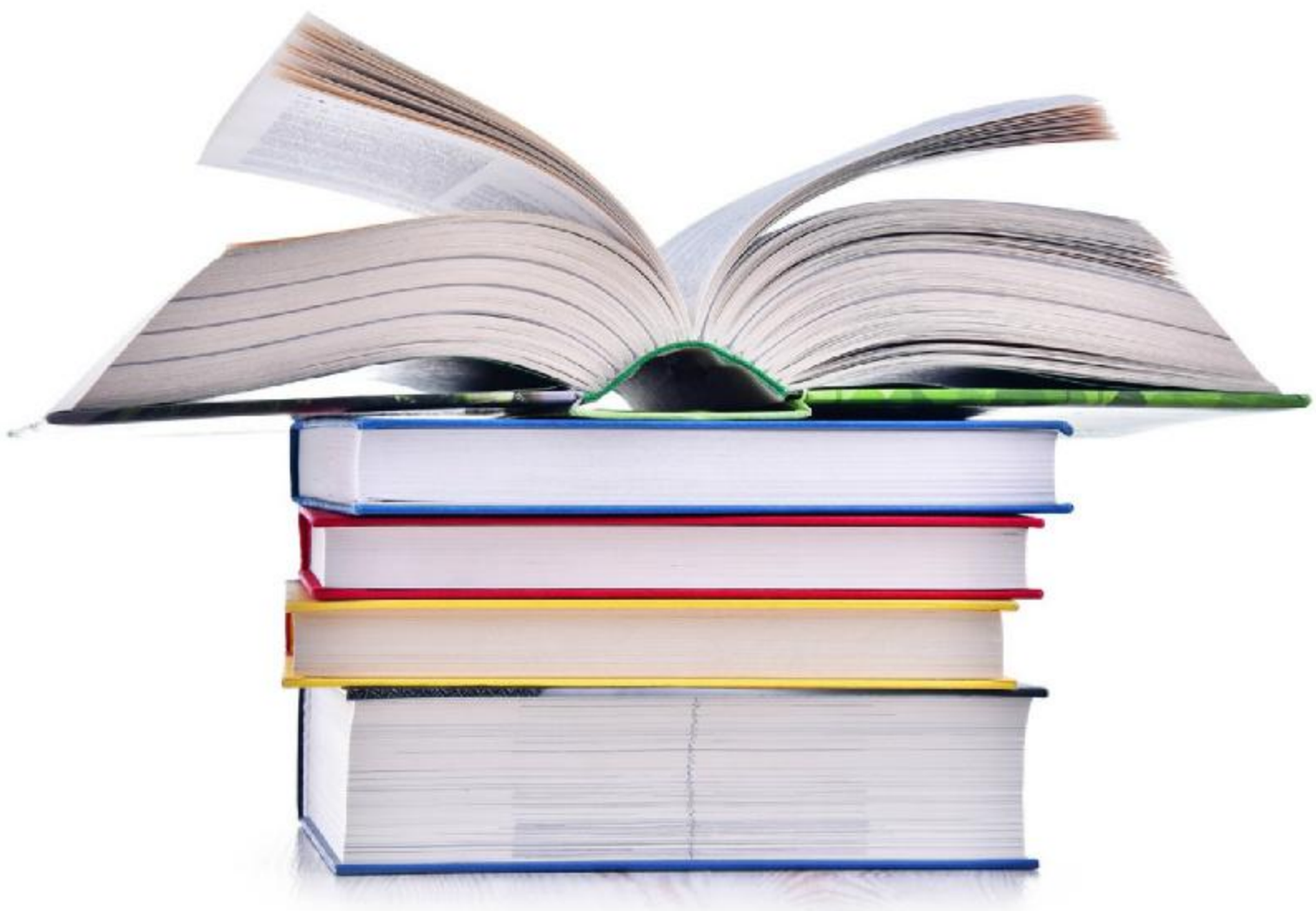
CONCLUSION

Nous voici arrivés à la fin de cet article. Il n'y a rien de révolutionnaire dans ce que je vous ai présenté, juste un peu de culture générale Python, qui pourra vous servir un jour pour écrire ou lire du code... ■

LES NOTES, RÉFÉRENCES ET GLOSSAIRES EN LATEX

TRISTAN COLOMBO

MOTS-CLÉS : LATEX, RÉFÉRENCES, NOTES, GLOSSAIRE



La gestion de la numérotation des notes, des références et la constitution de glossaires sont bien souvent un véritable cauchemar avec les traitements de texte. Avec LaTeX, tout cela est bien plus facile...

Il n'est pas rare de devoir enrichir un texte par des explications qui ne seront pas forcément nécessaires à l'ensemble des lecteurs. Pour cela, on dispose de plusieurs procédés :

- les notes de pied de page ou de marge contenant un texte explicatif ;
- les références qui permettent de citer un document, dont la description pourra être retrouvée dans une section « Bibliographie » ;
- le glossaire qui recense en fin de document les termes employés en indiquant la ou les pages dans lesquelles ils apparaissent.

Dans cet article, nous allons voir comment mettre en place ces éléments dans **LaTeX** et quels outils employer pour faciliter leur utilisation. Je considérerai que vous disposez déjà d'une version de LaTeX installée et que vous maîtrisez les bases de la création d'un document LaTeX **[1]**.

1. AVANT DE COMMENCER

Pour pouvoir montrer réellement la mise en place des éléments que sont les notes, les références et les glossaires, il faut disposer d'un document suffisamment imposant. Pour cela, j'avais deux solutions : soit employer un véritable document, soit en fabriquer un factice. J'ai retenu la seconde solution qui va nous permettre de découvrir deux nouveaux paquets. Le premier d'entre eux, **lipsum [2]**, dispose de 150 paragraphes de texte Lorem Ipsum auxquels on peut accéder pour les imprimer en sélectionnant lesquels (ce qui sera le premier paramètre de la commande `\lipsum`) et éventuellement, à l'intérieur de ces derniers, quelles phrases (second paramètre).

Voici un exemple d'utilisation :

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}

\usepackage{lipsum}

\begin{document}

  \section{Introduction}
  \lipsum[2-4]

  \section{Avant de commencer}
  \lipsum[2][3]

\end{document}
```

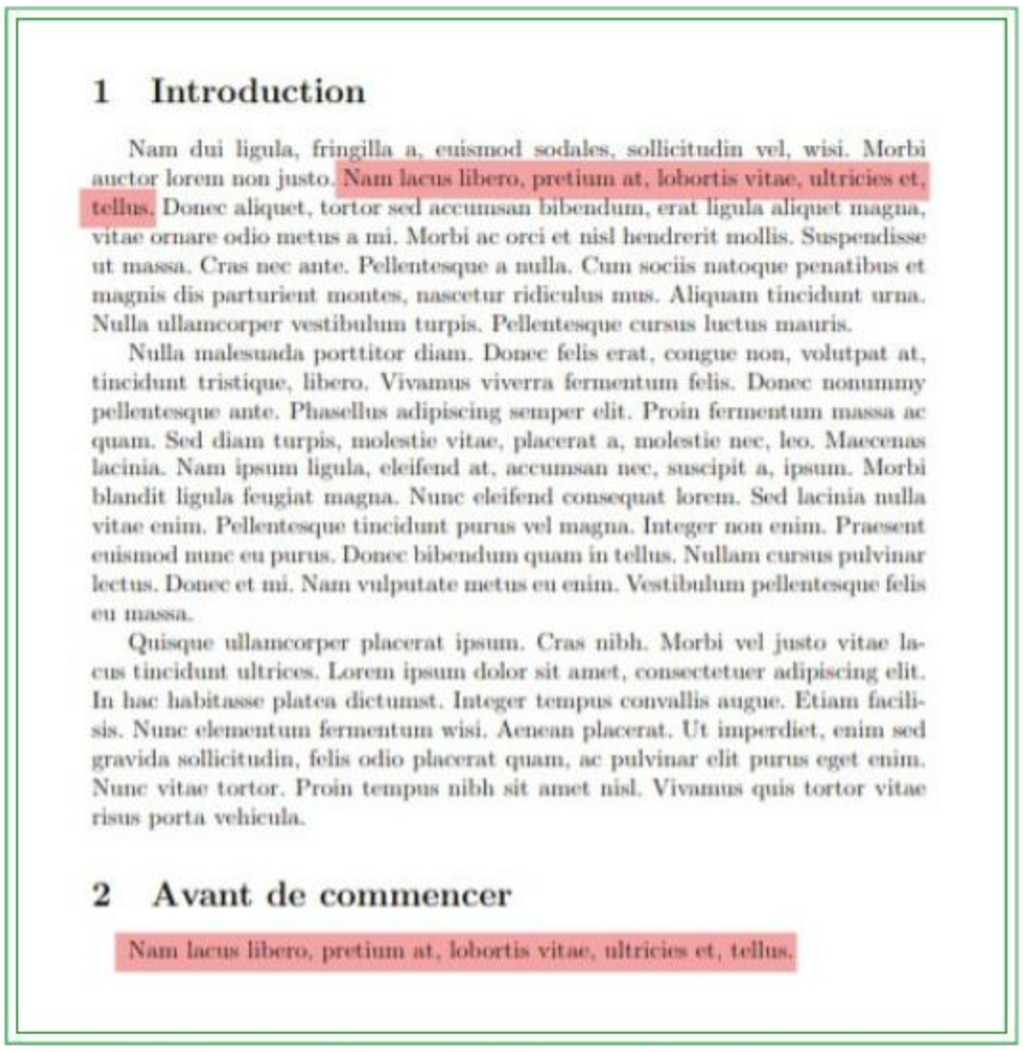
Le premier appel de `\lipsum` affiche les paragraphes 2 à 4 (`[2-4]`) et le second appel affiche la 3e phrase du second paragraphe (`[2][3]`). Le résultat de ce code est visible en figure 1.

Il existe un autre paquet permettant de remplir des zones de texte : **blindtext [3]**. Ce paquet permet de générer du contenu de différents types (paragraphes standard, mais également listes, formules mathématiques, etc.).

Voici un exemple d'utilisation :

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}

\usepackage{blindtext}

\begin{document}
```

```
  \blindtext[2]

  \blindmathtrue
  \blindtext

  \blinditemize[10]

\end{document}
```



1 Introduction

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

2 Avant de commencer

Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus.

Fig. 1 : Utilisation du paquet lipsum. On voit en rouge que la phrase de la section 2 correspond bien à la 3e phrase du premier paragraphe de la section 1.

`\blindtext[2]` génère `2` paragraphes de Lorem Ipsum, `\blindmathtrue` permet d'activer la présence de formules mathématiques dans les paragraphes (`\blindmathfalse` la désactive), et `\blinditemize[10]` génère une liste à puces de `10` éléments. La figure 2 montre le résultat obtenu pour ce code.

Il aurait été possible de générer tout un document avec la seule commande `\blinddocument` ou `\Blinddocument` (voir figure 3). Cette solution permet de se retrouver dans un contexte plus près de la réalité et c'est donc celle que j'ai retenue :

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}
\usepackage{blindtext}
```

```
\begin{document}

  \blinddocument

  Partie de code permettant de tester

  \blinddocument

  \dots

\end{document}
```

Qu'est que c'est?. C'est une phrase français avant le lorem ipsum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Qu'est que c'est?. C'est une phrase français avant le lorem ipsum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Qu'est que c'est?. C'est une phrase français avant le lorem ipsum. $\sin^2(\alpha) + \cos^2(\beta) = 1$. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

- Premier point dans une list
- Deuxième point dans une list
- Troisième point dans une list
- Quatrième point dans une list
- Cinquième point dans une list
- Sixième point dans une list
- Septième point dans une list
- Huitième point dans une list
- Neuvième point dans une list
- Dixième point dans une list

Fig. 2 : Document généré à l'aide du paquet blindtext en insérant des formules mathématiques dans les paragraphes (en rouge).

2.3.1 Example pour une list (4*description)

Premier point dans une list
Premier point dans une list
Premier point dans une list
Premier point dans une list
Deuxième point dans une list
Deuxième point dans une list
Deuxième point dans une list
Deuxième point dans une list
Partie de code permettant de tester

3 Titres de niveau 1 (section)

Qu'est que c'est?. C'est une phrase français avant le lorem ipsum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Fig. 3 : Document généré en utilisant \documentblind.

2. DES NOTES, DES NOTES, OUI, MAIS OÙ ?

L'ajout de notes peut se faire en pied de page ou dans la marge. Pour cela, il faudra utiliser deux syntaxes distinctes.

2.1 Notes en pied de page

L'ajout d'une note en pied de page se fait à l'aide de la commande **\footnote** qui prend en paramètre le contenu de la note :

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}

\usepackage{blindtext}

\begin{document}

  \blinddocument

  \section{Ajout de notes}
  Voici un exemple de note de bas de
page\footnote{Ceci est le contenu de la
note! On peut y ajouter des formules
mathématiques si on le souhaite :
```

```
$a^2-b^2 = (a-b)(a+b)$.} très simple.
  \blinddocument

  \dots

\end{document}
```

Notez que la note se rapporte au mot « page » et qu'il n'y a donc pas d'espace entre **page** et **\footnote**. Visuellement, cela ne changerait rien, le résultat serait le même qu'en figure 4, mais cela permet de rendre le code plus compréhensible.

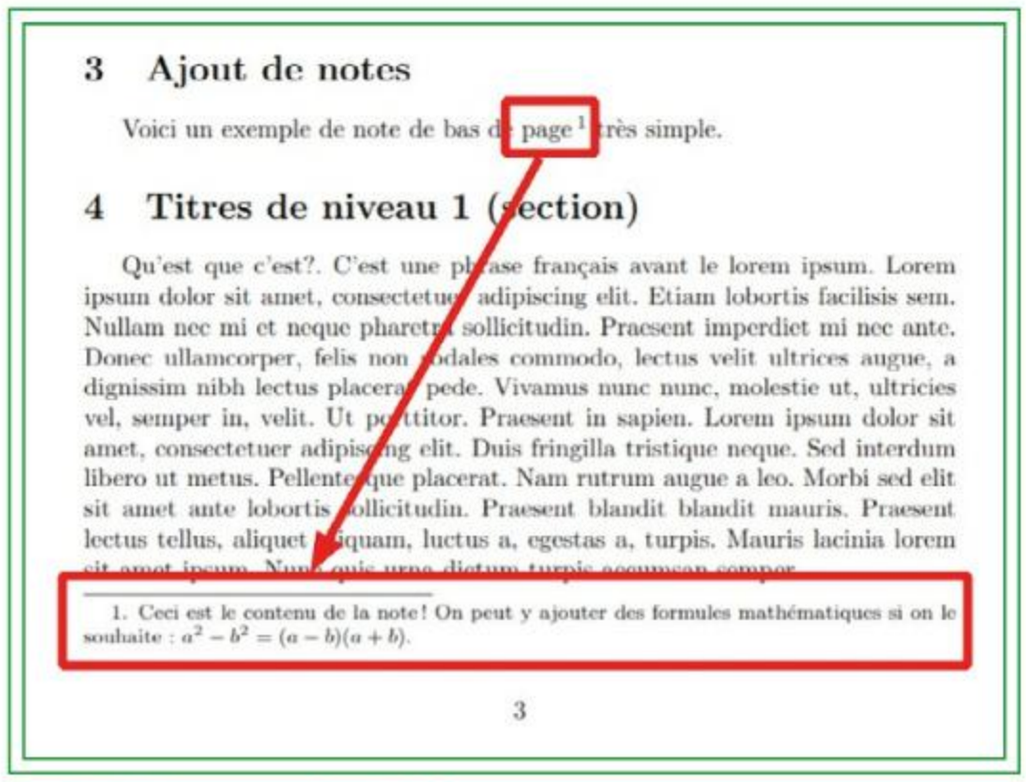


Fig. 4 : Une note de pied de page.

2.2 Notes dans la marge

La commande **\marginpar** permet d'ajouter des notes dans la marge. Elle s'utilise de la même manière que **\footnote** :

```
...
  \blinddocument

  \section{Ajout de notes}
  Voici un exemple de note de bas de
page\footnote{Ceci est le contenu de la
note! On peut y ajouter des formules
mathématiques si on le souhaite :
$a^2-b^2 = (a-b)(a+b)$.} très simple.
  Voici un exemple de note dans la
marge\marginpar{Note dans la marge avec
des maths : $\sqrt{a^2-b^2}$} grâce à
la commande \verb+\marginpar+.
...
```

Contrairement aux notes de bas de page, les notes dans la marge ne sont pas numérotées et commencent sur la ligne sur laquelle elles ont été définies (figure 5).

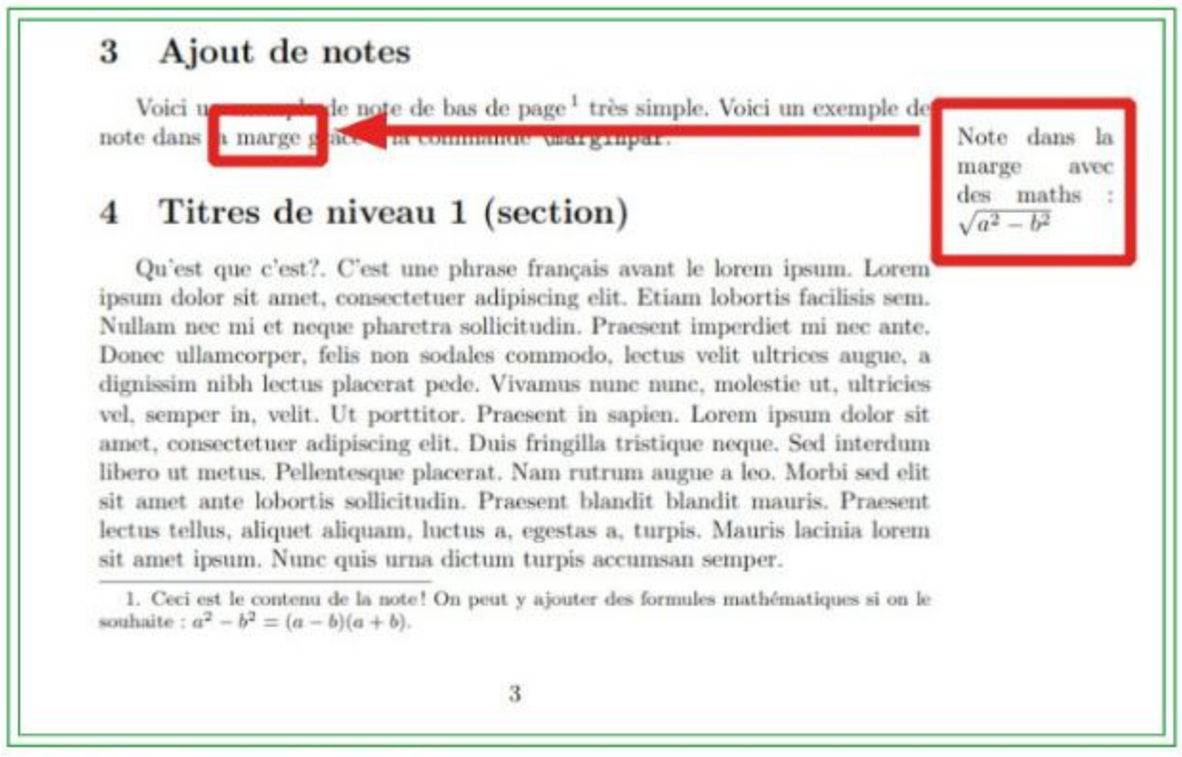


Fig. 5 : Une note dans la marge.

ATTENTION !

Si l'on ajoute des notes dans la marge de manière trop rapprochée, l'alignement ne sera plus respecté (figure 6) :

```
Voici un exemple de note dans la marge\
marginpar{Note dans la marge avec des
maths : $\sqrt{a^2-b^2}$} grâce à la
commande \verb+\marginpar+\marginpar{Et là
l'alignement n'est plus bon !}.
```

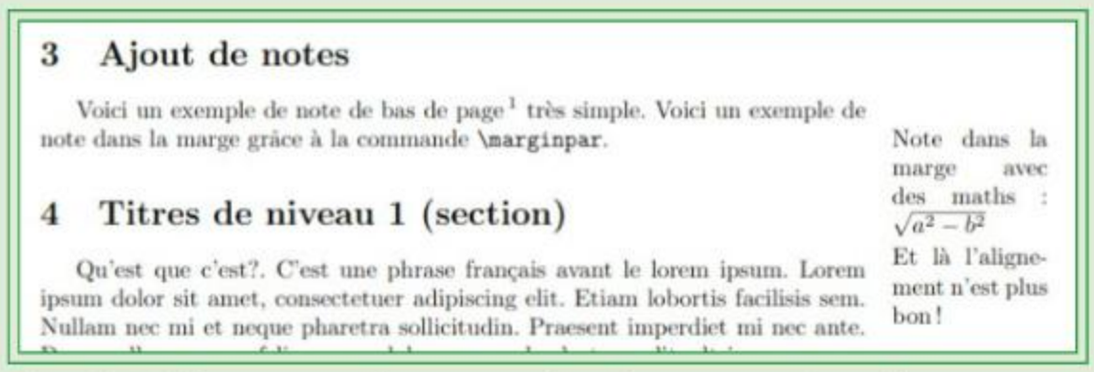


Fig. 6 : L'alignement des notes dans la marge n'est plus respecté, car les appels à \marginpar sont trop rapprochés.

3. LES RÉFÉRENCES

Lorsque l'on parle de références, on pense aussitôt aux références bibliographiques. Mais il y a en fait deux types de références : les références internes, où l'on souhaite indiquer dans quelle section ou à quelle page se trouve une information particulière et ensuite bien entendu, les références bibliographiques.

3.1 Références internes

Il est courant dans un document de vouloir faire référence à des informations se situant précédemment ou dans la suite. Pour indiquer au lecteur où trouver l'information, on lui indique alors soit le numéro de section, soit la page, soit encore les deux.

On commence par cibler le texte auquel on souhaite faire référence en insérant une étiquette à l'aide de `\label{<nom_référence>}`. La commande `\ref` permet alors d'afficher la section dans laquelle se trouve la référence dont le nom est passé en paramètre. Pour obtenir le numéro de page, on utilisera `\pageref` à la place :

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}

\usepackage{blindtext}

\begin{document}

  \blinddocument

  \section{Ajout de références}
  \label{target}
  Ici le texte auquel je vais faire
référence. On peut d'ailleurs trouver aussi
une référence en page~\pageref{ref_2}.

  \blinddocument

  \section{Fin du document}
  \label{ref_2}
  La référence a été créée en~\ref{target}
(on peut aussi mentionner la page en disant :
en~\ref{target}, page~\pageref{target}).

  \dots

\end{document}
```

Les étiquettes peuvent être placées n'importe où dans le document. Pour une meilleure lisibilité, je préfère les placer en début de section, mais c'est un choix personnel. Chaque appel à `\ref` ou `\pageref` est précédé d'une espace insécable (`~`) afin de ne pas autoriser de retour à la ligne avant l'affichage du numéro de section ou de la page.

ATTENTION !

L'utilisation des références nécessite une double compilation pour permettre à LaTeX d'enregistrer l'emplacement des étiquettes lors de la première compilation, puis de compléter le document avec les numéros de section ou de page lors de la seconde compilation.

Après une seule compilation, vous verrez apparaître des `??` aux emplacements des appels à `\ref` et `\pageref`, car les étiquettes ne sont pas encore connues.

Si vous souhaitez raffiner un peu l'affichage du texte indiquant où se trouve une référence, le paquet **varioref** permet d'afficher « de la présente page » si la référence pointée est présente sur la même page que l'appel ou encore « de la page ci-contre » pour la page précédente. Ce paquet fournit quatre nouvelles commandes :

- `\vref` : pour afficher le numéro de section ;
- `\vpageref` : pour afficher le numéro de page ;
- `\vrefrange` : pour afficher une référence à une zone délimitée par deux sections (la commande prend deux paramètres et affiche les numéros de section et de pages) ;
- `\vpagerefrange` : idem que la commande précédente, mais n'affiche que les numéros de page.

Voici un exemple d'utilisation de ces commandes :

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}

\usepackage{blindtext}
\usepackage[french]{varioref}

\begin{document}

  \blinddocument

  \section{Ajout de références}
  \label{target}
  Ici le texte auquel je vais faire
référence. On peut d'ailleurs trouver aussi
une référence en~\vpageref{ref_2}.
  Texte~\vpageref{target}.

  \blinddocument

  \section{Fin du document}
  \label{ref_2}
  La référence a été créée en~\vref{target}
(on peut aussi mentionner la page en disant :
voir~\vpageref{target}).\\
  Zone :~\vrefrange{target}{ref_2}.

  \dots

\end{document}
```

Si vous souhaitez créer des documents contenant des hyperliens comme dans une page web, le paquet **hyperref** est fait pour vous. Vous pourrez paramétrer simplement la couleur des liens internes et externes et montrer ainsi à vos lecteurs qu'il s'agit d'un lien cliquable. Voici un petit exemple d'utilisation et son résultat en figure 7.

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}

\usepackage{blindtext}
\usepackage[urlcolor=blue,
linkcolor=red, filecolor=cyan,
colorlinks=true]{hyperref}

\begin{document}

  \blinddocument

  \section{Ajout de références}
  Vous pouvez vous rendre directement
en fin de document en cliquant
\hyperlink{ref_2}{ici}.\\
  Pour visiter Connect, c'est sur~:
\url{https://connect.ed-diamond.com/}
que l'on peut écrire également
\href{https://connect.ed-diamond.com/}
{Connect}.\\
  Vous pouvez même créer un lien vers
un fichier : \href{run:./doc.log}{doc.
log}.


  \blinddocument

  \section{Fin du document}
  Cible du \hypertarget{ref_2}{lien}.


  \dots

\end{document}
```

Les couleurs des différents types de liens doivent être définies dans les options lors de l'import du paquet.

Pour créer un lien interne, cela se déroule en deux étapes :

1. On crée le lien cliquable en indiquant l'étiquette de la cible : **\hyperlink{<étiquette_cible>}{<texte_du_lien>}** ;
2. On indique où se trouve la cible : **\hypertarget{<étiquette_cible>}{<texte>}**.

NOTE

Pour avoir des liens qui se ciblent l'un l'autre (on clique sur le lien, puis on peut cliquer sur un autre lien pour revenir au point de départ), la structure du code devra être la suivante :

```
...
  \section{Ajout de références}
  Vous pouvez vous rendre directement en fin
de document en cliquant \hyperlink{target}
{\hypertarget{ref_2}{ici}}.\\

  ...

  \section{Fin du document}
  Cible du \hyperlink{ref_2}{\hypertarget
{target}{lien}}.
...
```

Pour créer un lien externe vers une page internet et afficher l'URL de cette page, il faudra employer la commande **\url** à laquelle on passera l'URL en paramètre. Pour « cacher » l'URL, on utilisera **\href** qui prend en premier paramètre l'URL, suivie en second paramètre du texte à afficher. Notez qu'il est possible de créer un lien vers un fichier en utilisant **\href** et en donnant pour URL **run:** suivi du chemin vers le fichier. Lorsqu'un utilisateur cliquera sur ce lien, le fichier s'ouvrira dans son éditeur configuré par défaut.

Enfin, sachez que si votre document contient une table des matières (**\tableofcontents**), au chargement du paquet **hyperref**, celle-ci deviendra automatiquement cliquable, permettant ainsi de se déplacer très simplement entre les différentes sections du document.

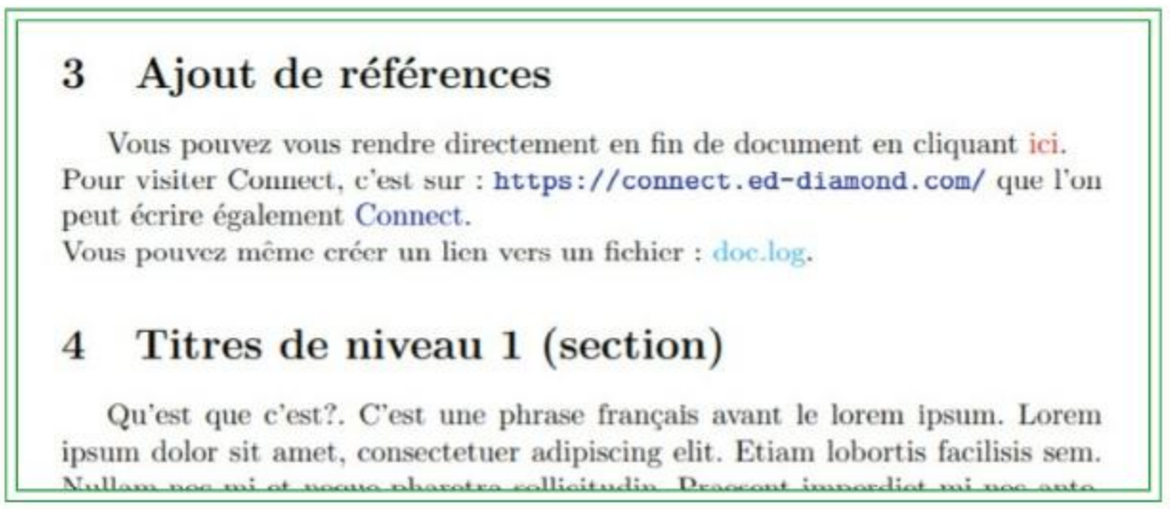


Fig. 7 : Aperçu de l'affichage de différents types de liens (paquet hyperref).

3.2 Références bibliographiques

3.2.1 BibTeX

Inclure des références bibliographiques manuellement dans un document n'est pas toujours une partie de plaisir :

- en fonction des documents, le format attendu n'est pas toujours le même tant au niveau de la citation que de la référence elle-même ;
- si les références sont numérotées par ordre d'apparition, l'insertion d'une nouvelle référence oubliée provoquera une renumérotation manuelle de toutes les références suivantes avec tous les risques d'erreurs qui en découlent ;
- les références doivent être ordonnées en fin de document (ordre d'apparition dans le document ou ordre alphabétique ou...) ;
- etc.

Heureusement que LaTeX, grâce au programme **BibTeX**, peut gérer tout cela pour nous.

NOTE

Il existe une méthode semi-automatisée faisant intervenir une liste spéciale **thebibliography**, mais je ne développerai pas son utilisation dans le présent article, car elle est à réserver aux bibliographies contenant un maximum de trois ou quatre références, BibTeX devenant rapidement beaucoup plus efficace et personnalisable.

Pour utiliser BibTeX, il faut disposer d'un ou plusieurs fichiers texte (extension **.bib**) contenant les références bibliographiques. Pour chaque entrée, il faut indiquer un type (article, livre, etc.) ainsi qu'une liste de champs obligatoires et éventuellement de champs optionnels.

Le tableau ci-dessous liste quelques-uns des types les plus employés, accompagnés des champs obligatoires et optionnels.

Prenons donc un fichier **biblio.bib** d'exemple (le premier champ de chaque entrée est l'identifiant et doit donc être unique) :

```
@article{
    LATEX_1,
    author = "Colombo, T",
    title = "Les bases de LaTeX sous GNU/Linux et
Windows",
    journal = "GNU/Linux Magazine France",
    year = "2020",
    month = "novembre",
    volume = "242",
    note = "\url{https://connect.ed-diamond.com/
GNU-Linux-Magazine/GLMF-242/Les-bases-de-LaTeX-
sous-GNU-Linux-et-Windows}"
}

@misc{
    LIPSUM,
    title = "Documentation du paquet lipsum",
    note = "\url{http://tug.ctan.org/tex-archive/
macros/latex/contrib/lipsum/lipsum.pdf}"
}

@misc{
    BINDTEXT,
    title = "Documentation du paquet bindtext",
    note = "\url{https://mirrors.ircam.fr/pub/
CTAN/macros/latex/contrib/blindtext/blindtext.pdf
}
```

Type	Cas d'utilisation	Champs obligatoires	Champs optionnels
@article	Article de journal	**author**, **journal**, **title**, **year**	**month**, **note**, **number**, **pages**, **volume**
@book	Livre	**author**, **publisher**, **title**, **year**	**address**, **edition**, **month**, **note**, **series**, **volume**
@conference	Article dans les actes d'une conférence	**author**, **booktitle**, **title**, **year**	**address**, **editor**, **month**, **note**, **organization**, **publisher**, **series**, **volume**
@misc	Autre...		**author**, **howpublished**, **month**, **note**, **title**, **year**

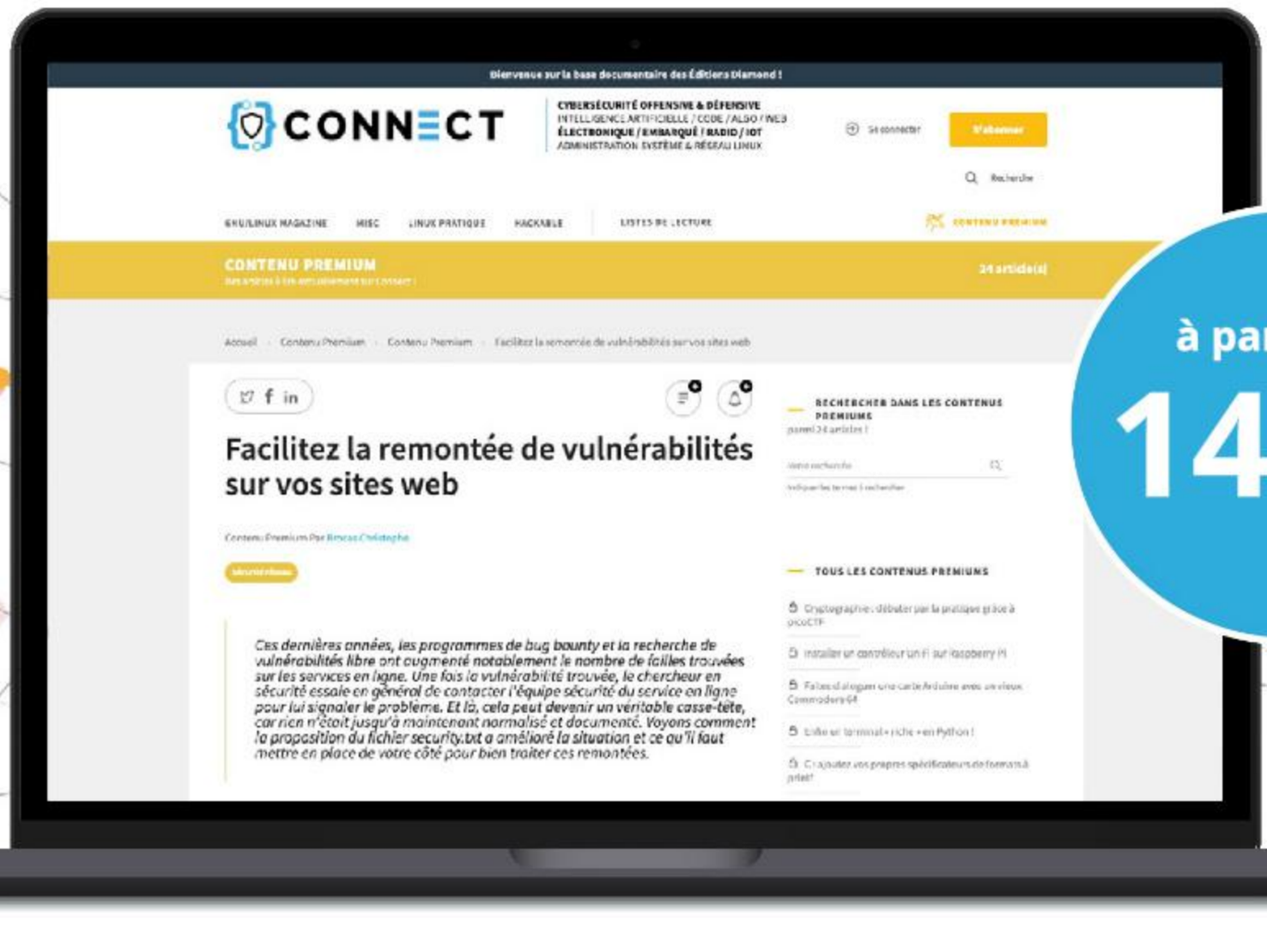
CONNECT

LA DOCUMENTATION TECHNIQUE DES PROS DE L'IT

Propulsé par

LINUX MAGAZINE / FRANCE · MISC · HACKABLE MAGAZINE · LINUX PRATIQUE

Le meilleur de l'information technique IT depuis 25 ans !



à partir de **149 € HT / an**

Contactez-nous pour une présentation

Service commercial · connect@ed-diamond.com · 03 67 10 00 20

connect.ed-diamond.com

Toujours disponible sur

www.ed-diamond.com

PAPIER

FLIPBOOK HTML5

sur www.ed-diamond.com

CONNECT

LA DOCUMENTATION TECHNIQUE DES PROS DE L'IT

sur connect.ed-diamond.com

Une fois la liste des références correctement structurée, il suffit de les citer dans le document à l'aide de `\cite{<identifiant>}`.

NOTE

Il est possible de citer plusieurs références en les séparant par des virgules : `\ref{<identifiant_1>, ..., <identifiant_n>}`.

NOTE

Pour afficher une référence dans la bibliographie sans la citer dans le texte (il y a quelque chose qui ne va pas dans votre document !), vous pouvez utiliser `\nocite{<identifiant>}` (ou plus simplement corriger votre document pour faire apparaître la citation !).

Il ne vous reste plus qu'à choisir un style de présentation pour les citations et les références, puis à afficher la bibliographie :

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}

\usepackage[urlcolor=blue,
citecolor=blue, colorlinks=true]
{hyperref}

\begin{document}

  Comme vu dans \cite{LATEX_1}, il est
possible d'utiliser des paquets
\cite{LIPSUM, BINDTEXT} pour étendre
les fonctionnalités de \LaTeX.

  \bibliographystyle{plain}
  \bibliography{biblio}

\end{document}
```

Ici, du fait du chargement d'`hyperref`, les références seront cliquables et on leur affecte donc une couleur spécifique (`citecolor=blue`).

Le style d'affichage choisi fait partie de l'un des quatre styles fournis par défaut :

- `plain` : entrées triées alphabétiquement et étiquetées par des numéros entre crochets (voir figure 8) ;
- `unsrt` : entrées triées par ordre d'apparition dans le document et étiquetées par des numéros entre crochets ;
- `alpha` : entrées triées alphabétiquement et étiquetées par le nom de l'auteur et l'année de parution de la référence entre crochets ;
- `abbrv` : comme `plain`, mais avec pour étiquettes les noms, noms de mois et noms de journaux abrégés entre crochets.

Comme vu dans [3], il est possible d'utiliser des paquets [2, 1] pour étendre les fonctionnalités de LaTeX.

Références

[1] Documentation du paquet bindtext. https://mirrors.ircam.fr/pub/CTAN/macros/latex/contrib/blindtext/blindtext.pdf.

[2] Documentation du paquet lipsum. http://tug.ctan.org/tex-archive/macros/latex/contrib/lipsum/lipsum.pdf.

[3] T Colombo. Les bases de latex sous gnu/linux et windows. *GNU/Linux Magazine France*, 242, novembre 2020. https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-242/Les-bases-de-LaTeX-sous-GNU-Linux-et-Windows.

Fig. 8 : Exemple de bibliographie avec le style plain.

Le simple fait de changer le style modifiera complètement l'affichage, comme vous pouvez le voir en figure 9 où `\bibliographystyle{plain}` a été remplacé par `\bibliographystyle{alpha}`.

Comme vu dans [Col20], il est possible d'utiliser des paquets [LIP, BIN] pour étendre les fonctionnalités de LaTeX.

Références

[BIN] Documentation du paquet bindtext. https://mirrors.ircam.fr/pub/CTAN/macros/latex/contrib/blindtext/blindtext.pdf.

[Col20] T Colombo. Les bases de latex sous gnu/linux et windows. *GNU/Linux Magazine France*, 242, novembre 2020. https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-242/Les-bases-de-LaTeX-sous-GNU-Linux-et-Windows.

[LIP] Documentation du paquet lipsum. http://tug.ctan.org/tex-archive/macros/latex/contrib/lipsum/lipsum.pdf.

Fig. 9 : Exemple de bibliographie avec le style alpha.

NOTE

Si vous utilisez **Visual Studio Code** et l'extension **LaTex Workshop [1]**, il vous suffira de sauvegarder votre document LaTeX pour générer la bibliographie. Sinon, vous devrez procéder à plusieurs compilations :

```
$ pdflatex doc
$ bibtex doc
$ pdflatex doc
$ pdflatex doc
```

Tous les journaux scientifiques proposent des fichiers de style LaTeX pour que les auteurs puissent fournir leurs articles en respectant le format attendu (https://template-selector.ieee.org/secure/templateSelector/publicationType pour **IEEE**, etc.). Les fichiers de style sont des fichiers d'extension **.bst** et il suffit de les placer dans le répertoire du document pour les utiliser. Par exemple, en téléchargeant **IEEEtranS.bst** et en le plaçant dans le répertoire de notre document de test, il suffit de faire pour l'utiliser :

```
...
  \bibliographystyle{IEEEtranS}
...
```

NOTE

Les fichiers de style **.bst** ont recours à une syntaxe particulière et expliquer comment créer un fichier de style personnalisé serait trop long pour cet article. Nous aborderons peut-être cela dans un autre article...

Vous concevrez que l'on peut difficilement faire plus souple comme système ! Par contre, la gestion des références bibliographiques dans des fichiers texte peut sembler être une pratique d'un autre âge... et c'est pour cela qu'il existe des logiciels de gestion de données bibliographiques !

3.2.2 Zotero

Zotero [4] est un logiciel de gestion de données bibliographiques auquel on peut adjoindre un connecteur afin de récolter automatiquement les références des articles que l'on consulte dans notre navigateur (pour peu que le site en question soit compatible, sinon Zotero essaye de faire comme il peut pour retrouver les données).

Je passerai rapidement sur la partie installation, puisque tout est détaillé dans **[4]** :

1. On télécharge la dernière version sur https://www.zotero.org/download/ ;
2. On installe le programme :

   → Sous **GNU/Linux** :

```
$ tar -xvjf Zotero-5.0.96_linux-x86_64.tar.bz2 -C /opt
```

   → Sous **Windows** :

   On lance **Zotero-5.0.96_setup.exe** et on suit les instructions du mode graphique.

3. Lancez **zotero** :

   → en ligne de commande pour GNU/Linux :

```
$ cd opt/Zotero-5.0.96_linux-x86_64
$ zotero
```

   → en cliquant dans le menu pour Windows.

Au premier démarrage, Zotero vous propose automatiquement l'installation du connecteur pour **Chrome** (si vous utilisez ce dernier). Votre navigateur ouvrira la page https://www.zotero.org/start et vous n'aurez qu'à cliquer sur ***Install***, puis ***Ajouter à Chrome***.

Pour récupérer les données du fichier **biblio.bib** créé précédemment, il suffit de cliquer dans l'application Zotero sur ***Fichier > Importer...*** et de sélectionner le fichier **biblio.bib**.

La figure 10 montre les données de **biblio.bib** dans l'interface graphique de Zotero. Pour les modifier ou ajouter des données, tout peut désormais passer par le mode graphique, ce qui est beaucoup plus simple et convivial que d'utiliser un fichier texte !

Mais... LaTeX attend un fichier **.bib** ! Comment le lui fournir ?

Cliquez sur ***Fichier > Exporter la bibliothèque...*** et sélectionnez le format BibTeX (figure 11) : le tour est joué !

4. LES GLOSSAIRES... ET PLUS !

Le principe d'un glossaire est de définir des termes spécifiques qui peuvent perturber le lecteur. Il est également possible de définir les acronymes d'un document.

Attention, il ne faut pas confondre le glossaire qui contient des définitions et l'index qui lui recense seulement les pages où les termes apparaissent. Dans cette section, nous aborderons les trois cas énoncés : index, définition d'acronymes et glossaire.

4.1 Index

Construire un index n'est pas très compliqué : il suffit d'utiliser le paquet **makeidx** et d'indiquer les termes à ajouter dans l'index à l'aide de la commande **\index**. La syntaxe employée pour passer le paramètre attendu à **\index** est un peu particulière :

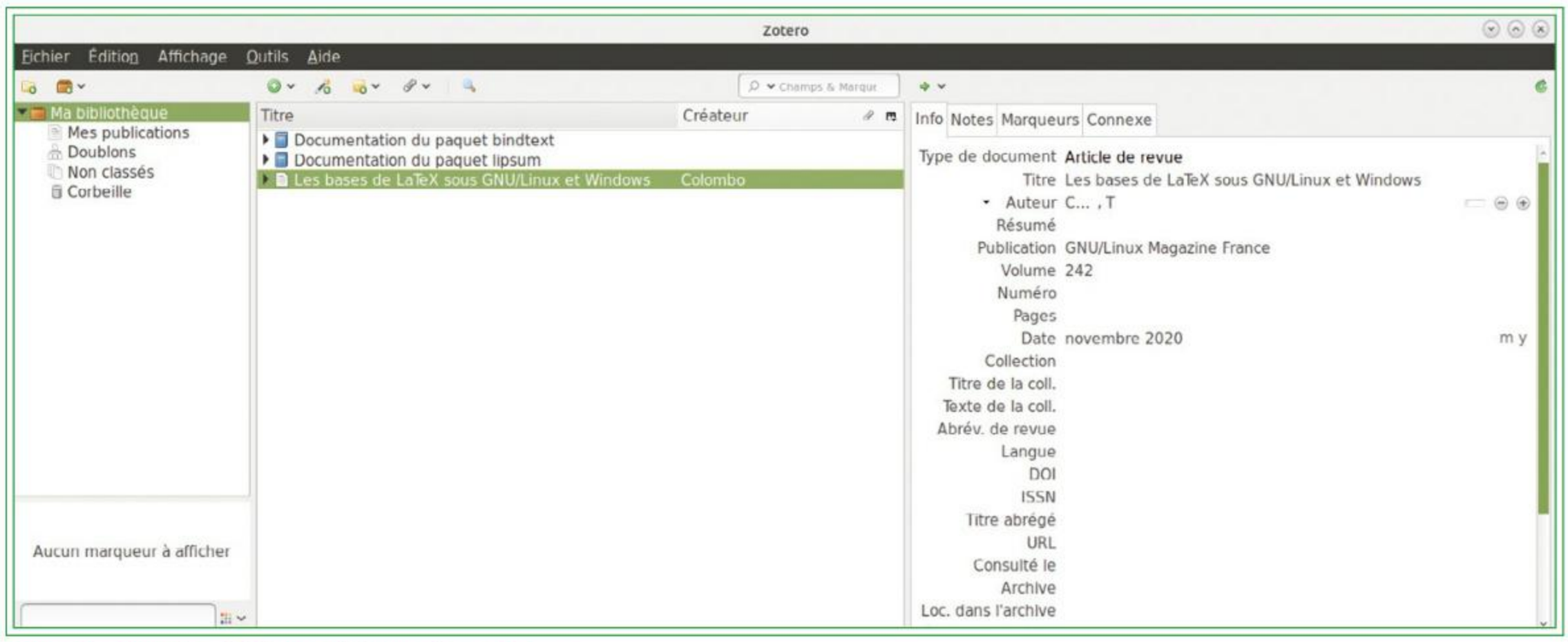


Fig. 10 : Données bibliographiques dans Zotero.

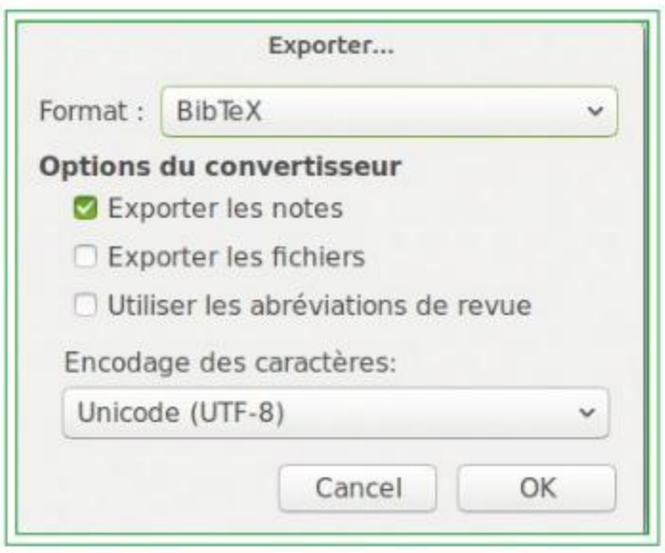


Fig. 11 : Export de données depuis Zotero au format BibTeX.

- `\index{<mot>}` : référence `<mot>` dans l'index sous la forme `mot, x` où `x` est un numéro de page dans laquelle apparaît `mot` ;
- `\index{<mot>@<affiche>}` : référence de `<affiche>` dans l'index en utilisant `<mot>` pour effectuer le tri alphabétique déterminant l'ordre des mots dans l'index. Ceci est dû au fait que LaTeX ne sait pas ordonner les caractères accentués correctement et permet donc de compenser cette lacune ;
- `\index{<mot>!<groupe>}` : référence `<mot>` dans l'index comme sous élément de `<groupe>`. Ceci permet de grouper des définitions sous une même entrée principale ;
- `\index{<mot>|<cmd>}` : référence `<mot>` dans l'index en suivant les instructions de formatage `<cmd>`. Ces instructions peuvent être :
  - → `see{<autre_mot>}` qui provoquera l'affichage de voir `<autre_mot>` au lieu du numéro de page ;
  - → `textbf` ou `textit` pour afficher le numéro de page en gras ou en italique ;
  - → `(` et `)` respectivement pour démarrer un plage de pages et pour la terminer.

ATTENTION !

N'ajoutez jamais d'espace avant ou après les caractères `@`, `!` et `|` sous peine de ne pas obtenir le résultat attendu...

Voici un exemple utilisant cette syntaxe :

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}

\usepackage{blindtext}
\usepackage{makeidx}
\makeindex
```

```
\begin{document}

  \blinddocument

  \section{Ajout de liens}
  Voici un texte permettant d'ajouter des entrées dans
l'index\index{index}.
  Nous verrons ainsi plusieurs exemples de syntaxe
\index{index!syntaxe}.
  Pour ce qui est du glossaire\index{glossaire|(}, nous
verrons cela plus tard.
  Et enfin un test sur les mots en "e"~:
  \begin{itemize}
    \item éléphant\index{elephant@éléphant}~;
    \item énergie\index{energie@énergie}~;
    \item eau\index{eau|textbf}~;
    \item elle\index{elle}~;
  \end{itemize}

  \blinddocument

  \section{Fin du document}
  Toujours pas de glossaire\index{glossaire|)}~!

  \dots

  \printindex

\end{document}
```

Nous obtenons le résultat de la figure 12. Notez qu'en remplaçant `\index{elephant@éléphant}` par `\index{éléphant}`, l'ordre alphabétique ne sera plus respecté (figure 13).

4.2 Acronymes

Les acronymes sont gérés par le paquet **glossaries**, le même qui sera employé pour les glossaires, mais auquel on passe en option le terme `acronym` :

```
\usepackage[acronym]{glossaries}

\makeglossaries
```

Les acronymes doivent être définis en début de document en indiquant un identifiant, l'acronyme et sa signification. Prenons l'exemple de `GLMF` pour `GNU/Linux Magazine France`. Il faudra définir :

```
\newacronym{glmf}{GLMF}{GNU/Linux Magazine France}
```

`glmf` est l'identifiant, `GLMF` est l'acronyme et ensuite il s'agit de la définition.

Nous disposons ensuite de trois commandes pour afficher un acronyme dans un texte tout en le référençant dans la table finale des acronymes :

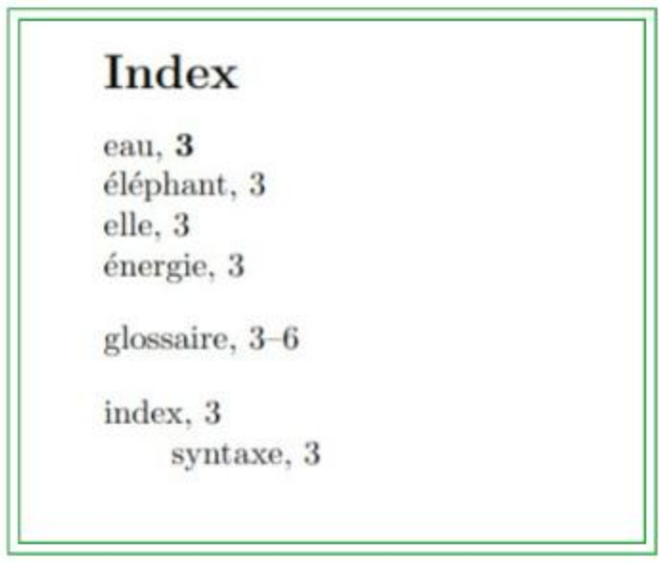


Fig. 12 : Exemple d'index.

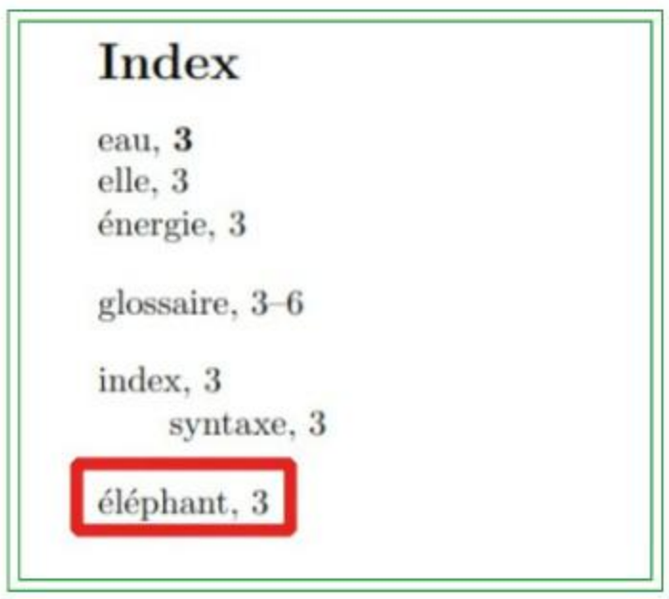


Fig. 13 : Index mal formé à cause d'une entrée accentuée non gérée avec @.

- `\acrlong{<identifiant>}` : affiche la définition de l'acronyme ;
- `\acrshort{<identifiant>}` : affiche l'acronyme ;
- `\acrfull{<identifiant>}` : affiche la définition de l'acronyme suivie de l'acronyme entre parenthèses.

Voyons cela dans un exemple :

```
\documentclass{article}

\usepackage[utf8]
{inputenc}
\usepackage[french]
{babel}

\usepackage{blindtext}
\usepackage[acronym]
{glossaries}
\makeglossaries
```

```
\newacronym{glmf}{GLMF}{GNU/Linux
Magazine France}
\newacronym{lp}{LP}{Linux Pratique}
\newacronym{hk}{HK}{Hackable}

\begin{document}

  \blinddocument

  \section{Ajout d'acronymes}
  Ceci est \acrshort{glmf} mais il existe
aussi \acrlong{lp} et \acrfull{hk}.


  \blinddocument

  \dots

  \clearpage
  \printglossary[type=\acronymtype]

\end{document}
```

Au niveau de l'apparition des acronymes, cela donnera le résultat de la figure 14 et la table des acronymes est visible en figure 15. Pour obtenir cette table sans l'extension LaTeX Workshop de Visual Studio Code, il faut lancer :

```
$ pdflatex doc
$ makeglossaries doc
$ pdflatex doc
$ pdflatex doc
```

3 Ajout d'acronymes

Ceci est GLMF mais il existe aussi Linux Pratique et Hackable (HK).

4 Titres de niveau 1 (section)

Qu'est que c'est?. C'est une phrase francais avant le lorem ipsum. Lorem

Fig. 14 : Différents types d'affichage des acronymes.

Acronymes

GLMF GNU/Linux Magazine France. 3

HK Hackable. 3

LP Linux Pratique. 3

Fig. 15 : Table des acronymes.

Et avec LaTeX Workshop... ça ne fonctionne pas tout seul ! Il va y avoir un peu de travail à faire au niveau du fichier **settings.json**. Éditez le fichier utilisateur en appuyant sur **<Ctrl>** + **<Shift>** + **<P>** et en tapant **open settings**, puis en choisissant ***Open Settings (JSON)***. Ajoutez les lignes suivantes :

```
{
    ...,

    // LATEX WORKSHOP
    // Recipes
    "latex-workshop.latex.recipes": [
        {
            "name": "Compile with glossaries",
            "tools": [
               "pdflatex",
                "makeglossaries",
                "pdflatex",
                "pdflatex"
            ]
        },
        {
            "name": "pdflatex",
            "tools":[
                "pdflatex",
            ]
        },
    ],
    // Tools
    "latex-workshop.latex.tools": [
        {
            "name": "pdflatex",
            "command": "pdflatex",
            "args": [
                "-synctex=1",
                "-interaction=nonstopmode",
                "-file-line-error",
                "%DOC%"
            ]
        },
        {
            "name": "makeglossaries",
            "command": "makeglossaries",
            "args": [
              "%DOCFILE%"
            ]
          }
    ],
}
```

Nous avons créé ici une nouvelle « recette » (**recipes**) pour l'extension LaTeX Workshop (**latex-workshop**). Cette nouvelle recette se nomme **Compile with glossaries** et va exécuter les « outils » (**tools**) **pdflatex** et **makeglossaries** dont on retrouve la définition dans les lignes suivantes. Nous utilisons ici deux variables **%DOC%** et **%DOCFILE%**. Les variables utilisables dans la section **tools** sont les suivantes :

- **%DOC%** : le chemin absolu du fichier LaTeX sans l'extension **.tex** ;
- **%DOCFILE%** : le nom du fichier LaTeX sans l'extension **.tex** ;
- **%DIR%** : le chemin absolu vers le répertoire contenant le fichier LaTeX ;
- **%TMPDIR%** : le nom du répertoire temporaire ;
- **%OUTDIR%** : le nom du répertoire de sortie (configuré dans **latex-workshop.latex.outDir**).

Rechargez la fenêtre pour activer les modifications (**<Ctrl>** + **<Shift>** + **<P>**, puis tapez **Reload Window**). La nouvelle recette apparaîtra dans le menu ***TeX*** (figure 16) et vous n'aurez plus qu'à l'utiliser.

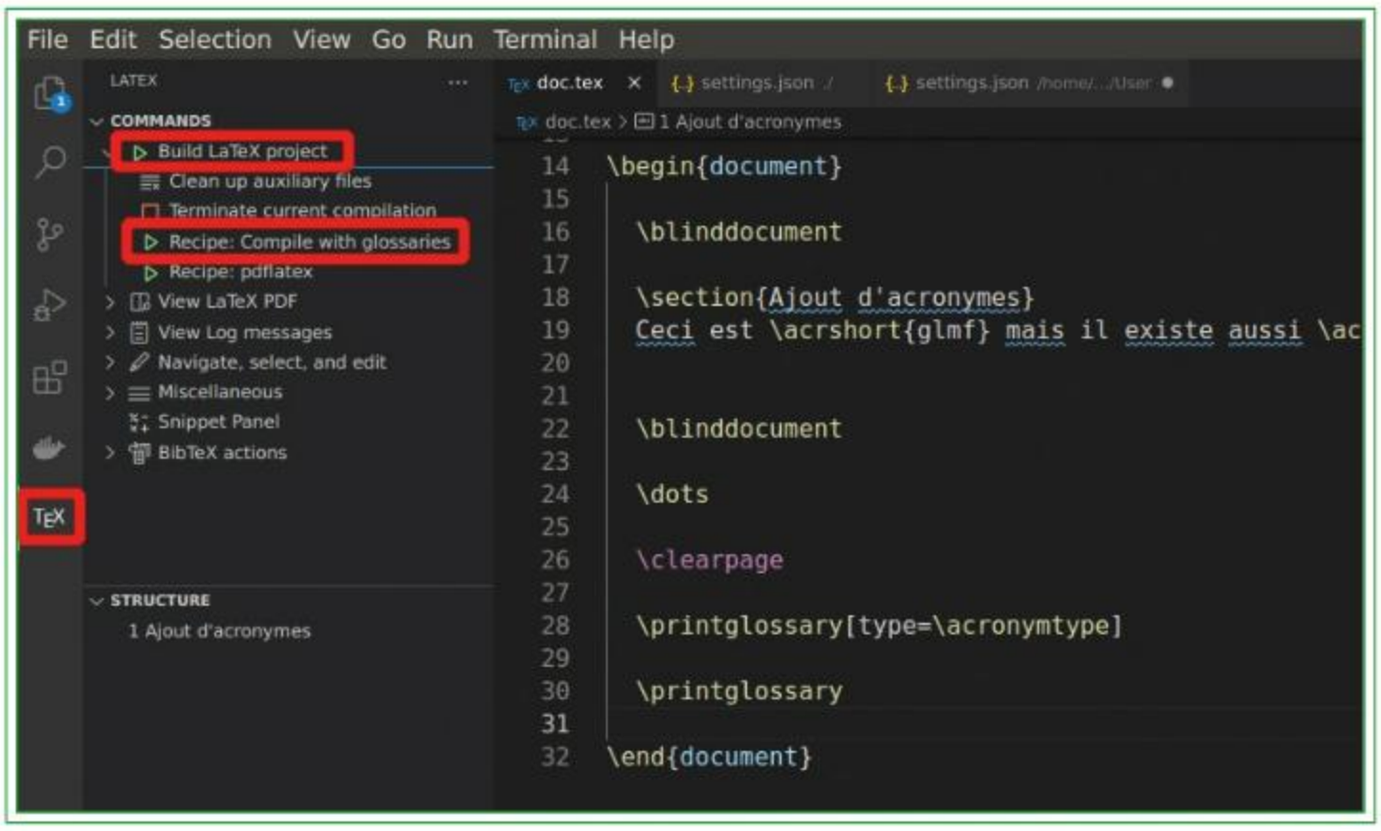


Fig. 16 : Recette « Compile with glossaries » ajoutée à l'extension LaTeX Workshop de Visual Studio Code.

NOTE

Pour un affichage de la table des acronymes dans la table des matières, ajoutez l'option **toc** au paquet :

```
\usepackage[acronym, toc]
{glossaries}
```

Avant d'en finir avec les acronymes, on peut se demander s'il ne serait pas possible de définir un fichier contenant une liste d'acronymes et de le charger au besoin. Cela permettrait d'éviter d'avoir à redéfinir inutilement des acronymes très utilisés.

La commande **\loadglsentries** prend en paramètre le nom d'un fichier LaTeX contenant des définitions. On peut ainsi imaginer un fichier **diamond.tex** contenant :

```
\newacronym{glmf}{GLMF}{GNU/Linux Magazine France}

\newacronym{lp}{LP}{Linux Pratique}

\newacronym{hk}{HK}{Hackable}
```

L'utilisation de ces définitions dans le fichier **doc.tex** se fera alors par :

```
\documentclass{article}

...

\usepackage[acronym]{glossaries}
\makeglossaries

\loadglsentries{diamond}

\begin{document}

  \blinddocument

  \section{Ajout d'acronymes}
  Ceci est \acrshort{glmf} mais il existe aussi
\acrlong{lp} et \acrfull{hk}.

  ...

  \printglossary[type=\acronymtype]

\end{document}
```

NOTE

On peut bien entendu imaginer de multiples fichiers, de manière à classer les différentes acronymes.

4.3 Glossaires

Après avoir vu comment gérer les acronymes, l'ajout de termes et l'impression d'un glossaire sont très semblables. Il n'y a que deux différences :

- la définition des termes se fait par `\newglossaryentry` en indiquant :
  - → l'identifiant ;
  - → le nom à définir ;
  - → la définition.
- l'utilisation d'un terme à référencer dans le glossaire se fait à l'aide de l'une des commandes suivantes :
  - → `\gls` : affiche le terme en minuscules ;
  - → `\Gls` : affiche le terme en minuscules avec la première lettre en majuscule ;
  - → `\glspl` et `\Glspl` : affichage au pluriel (avec un « s » à la fin) correspondant à `\gls` et `\Gls`.

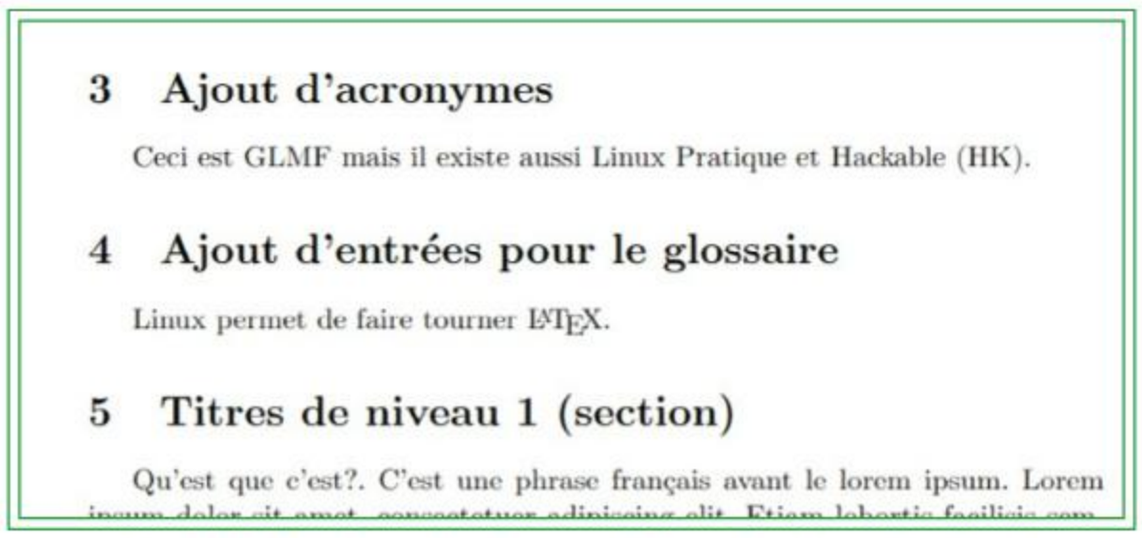
Voyons un exemple d'application :

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[french]{babel}

\usepackage{blindtext}
\usepackage[acronym]{glossaries}
\makeglossaries

\loadglsentries{diamond}

\newglossaryentry{linux}
{
    name=linux,
    description={Système d'exploitation
Open Source}
}
\newglossaryentry{latex}
{
    name=\LaTeX,
    description={Système de composition
de documents}
}
```

```
\begin{document}

  \blinddocument

  \section{Ajout d'acronymes}
  Ceci est \acrshort{glmf} mais il existe aussi
\acrlong{lp} et \acrfull{hk}.

  \section {Ajout d'entrées pour le glossaire}
  \Gls{linux} permet de faire tourner \gls{latex}.

  \blinddocument

  \dots

  \clearpage

  \printglossary[type=\acronymtype]

  \clearpage

  \printglossary

\end{document}
```

Les définitions de termes pourraient être regroupées dans un fichier externe comme nous l'avons fait précédemment pour les acronymes. Le code précédent va permettre d'afficher les termes dans le texte (figure 17) et de les référencer dans le glossaire (figure 18).

La compilation se fait de la même manière que pour les acronymes.



3 Ajout d'acronymes

Ceci est GLMF mais il existe aussi Linux Pratique et Hackable (HK).

4 Ajout d'entrées pour le glossaire

Linux permet de faire tourner LaTeX.

5 Titres de niveau 1 (section)

Qu'est que c'est?. C'est une phrase français avant le lorem ipsum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem.

Fig. 17 : Affichage de termes référencés dans le glossaire.

Glossaire

LaTeX Système de composition de documents. 3

linux Système d'exploitation Open Source. 3

Fig. 18 : Affichage du glossaire.

NOTE

Vous avez pu voir comment gérer l'affichage des acronymes et du glossaire de manière distincte :

```
\printglossary[type=\acronymtype]
% Acronymes
\printglossary     % Glossaire
```

Il est possible de tout afficher avec une seule commande :

```
\printglossaries
```

CONCLUSION

Ainsi s'achève ce nouvel article sur LaTeX. Petit à petit, nous découvrons comment réaliser des documents de plus en plus sophistiqués et précis. Il nous reste encore beaucoup de choses à apprendre avant de nous lancer dans la création de nos propres outils ! Mais d'ici là, vous devriez déjà être capable d'écrire un livre avec références bibliographiques et glossaire ! :-) ■

RÉFÉRENCES

[1] T. COLOMBO, « Les bases de LaTeX sous GNU/Linux et Windows », GNU/Linux Magazine n°242, novembre 2020 : https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-242/Les-bases-de-LaTeX-sous-GNU-Linux-et-Windows

[2] Documentation du paquet lipsum : http://tug.ctan.org/tex-archive/macros/latex/contrib/lipsum/lipsum.pdf

[3] Documentation du paquet blindtext : https://mirrors.ircam.fr/pub/CTAN/macros/latex/contrib/blindtext/blindtext.pdf

[4] V. MAGNIN, « Gérez votre bibliographie avec Zotero », Linux Pratique n°107, mai 2018 : https://connect.ed-diamond.com/Linux-Pratique/LP-107/Gerez-votre-bibliographie-avec-Zotero

FRAIS DE PORTS OFFERTS*

sur tous nos magazines en kiosque !

Pas envie de vous déplacer ? Profitez-en !

LIVRÉS CHEZ VOUS

GRATUITEMENT !*

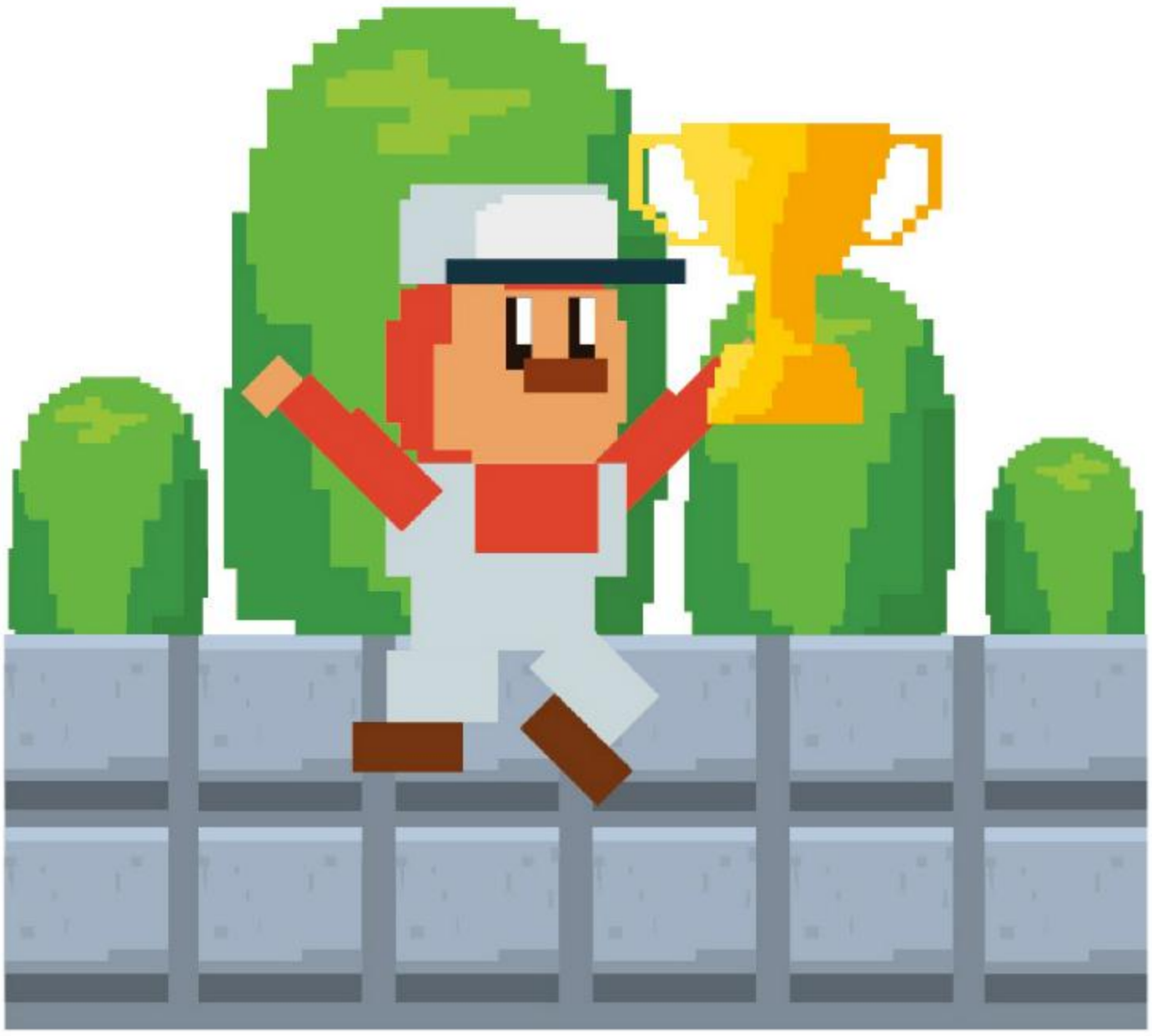
* Frais de ports offerts pour toute livraison en France métropolitaine, offre valable uniquement pour toutes les publications de la rubrique « En kiosque » de la boutique www.ed-diamond.com, appliquée une fois connecté à votre espace personnel.

Rendez-vous sur www.ed-diamond.com

ANALYSONS ET TROUVONS LES SOLUTIONS TECHNIQUES À NOS PROBLÉMATIQUES DE JEU GODOT

MICHAËL BERTOCCHI
[Ingénieur développement- EPEXSPOT, créateur du mkframework (framework PHP)]

MOTS-CLÉS : GDSCRIPT, GODOT, SYNTAXE



Dans cette série d'articles, nous allons découvrir comment créer un jeu aussi complexe qu'un jeu d'aventure avec le moteur de jeu Godot. Dans cette partie, nous verrons de nouveaux points techniques à résoudre.

Dans l'article précédent [1], nous avons pu créer le village, des habitations, le joueur et nous avons vu comment le faire se déplacer à l'écran, le faire entrer/sortir d'une habitation.

Le problème rencontré précédemment était le suivant : quand on entre puis on sort d'une habitation, on atterrit toujours au même endroit dans le village, peu importe l'habitation dans laquelle on était entré.

Nous allons voir dans cet article comment résoudre cette incohérence et nous verrons également comment gérer un solde de gems (pierres précieuses) et notre inventaire, puis comment gérer une boutique avec affichage dynamique d'objets achetables. Et enfin, nous verrons comment créer un contrôleur de jeu tactile.

NOTE

Le code présenté ici a été développé sous **Linux Mint**.

Le code source de ce projet est également disponible sur **GitHub** (branche **partie2**) : https://github.com/imikado/articleLMGodotLittleAdventure.

1. SINGLETON OU COMMENT GÉRER DES DONNÉES TRANSVERSALES

1.1 Gérer la position de notre joueur

Dans notre village, nous allons avoir plusieurs habitations, notamment la maison du joueur et la boutique. Nous souhaiterions, en sortant de chacune d'entre elles, que le joueur apparaisse en face de la porte en question. Pour cela, nous allons utiliser une fonctionnalité de Godot : les **singletons** (chargement automatique), l'idée étant qu'à chaque chargement de scène, nos variables initiées dans la scène précédente aient toujours le même statut.

Nous allons commencer par créer un fichier qui sera chargé en permanence : toutes les variables que vous y stockerez perdureront pendant toute la partie.

Créez les répertoires **common/singletons**, puis effectuez un clic droit sur ***new script***, laissez les paramètres par défaut et saisissez comme nom **globalPlayer.gd**.

Dans ***projet settings > onglet autoload***, cliquez sur le bouton « répertoire » pour sélectionner le fichier à charger, sélectionnez notre fichier **common/singletons/globalPlayer.gd**, puis validez avec le bouton ***add***.

Vous voyez dans le tableau du bas une ligne avec notre fichier : il sera donc instancié en permanence (notez la première colonne qui indique le nom sous lequel vous pourrez accéder à votre fichier, ici **GlobalPlayer**).

Éditons-le pour ajouter le code permettant de charger/stocker notre position.

Fichier **common/singletons/globalPlayer.gd** :

```
extends Node

var position=null

func _ready():
 pass # Replace with function body.

func savePosition(newPosition_):
 position=newPosition_

func loadPosition():
 var savedPosition=position
 resetPosition()
 return savedPosition

func resetPosition():
 position=null

func shouldLoadPosition():
 if(position!=null):
  return true
 else:
  return false
```

Ici, rien de bien compliqué : une méthode pour sauvegarder, une pour vérifier si on a une position sauvegardée, et une méthode pour récupérer la position pour la supprimer ensuite.

Maintenant que nous avons un moyen de stocker et de restituer la position du joueur, nous allons l'ajouter dans notre projet. Éditez d'abord le code du village dans le fichier **screens/tree-village.gd** :

```
extends Node2D

func getPlayer():
 return $YSort/player

func _ready():
 if(GlobalPlayer.shouldLoadPosition()):
  getPlayer().position=GlobalPlayer.loadPosition()
```

```
func _on_shop_playerEntered():
 var PositionToSave=getPlayer().position
 PositionToSave.y+=10
 GlobalPlayer.savePosition(PositionToSave)
 get_tree().change_scene("res://screens/tree-
village/shop.tscn")
```

Au moment d'entrer dans l'habitation, on sauve la position du joueur avec un décalage vertical de 10 pour éviter en revenant de réactiver la collision (et ainsi de boucler en continu).

Ensuite, on indique qu'au chargement de la scène, si on a une position sauvegardée, il faut forcer la position du joueur.

J'ai pour habitude de créer des méthodes d'accès à certains éléments utilisés, permettant ainsi de les déplacer dans l'arborescence plus tard, en ayant juste à mettre à jour ces méthodes d'accès.

1.2 Gérer les « gems »

Les gems, que l'on peut traduire par pierres précieuses, seront ici comme dans beaucoup d'autres jeux notre monnaie d'échange. Notre héros devra s'en procurer pour acheter des objets et des armes à d'autres personnages.

Nous aurons donc besoin de les stocker, connaître leur solde, en ajouter, et en dépenser.

Nous allons modifier notre script créé précédemment **common/singletons/globalPlayer.gd** :

```
...

var gems=0

...

#--gems
func putGems(value_):
 gems+=value_

func spendGems(value_):
 if(value_<gems):
  gems+=value_

func getGemsBalance():
 return gems

func canSpendGems(price_):
 if(price_<=gems):
  return true
 return false
```

Ici, on peut ajouter des gems, vérifier si on peut dépenser une somme, la dépenser et récupérer le solde. Toutes ces méthodes seront bien pratiques lors de l'entrée du joueur dans un magasin.

Mais avant cela, nous allons les afficher à l'écran pour que le joueur puisse voir en permanence son solde.

1.3 Gérer les objets et l'inventaire

Avant de parler de stockage et d'achat, nous devons d'abord créer ces différents objets.

Nous allons commencer par créer une classe qui permettra de définir un objet. Dans **common/**, créez un répertoire **class** et ajoutez-y un script **item_class.gd** avec en parent non pas **Node**, mais **Resource**. Éditez ensuite ce fichier **common/class/item_class.gd** :

```
extends Resource

class_name item_class

enum TYPE {WEAPON, MAGICPOTION }

var name
var description
var image
var type
var actionList

func _init(name_,description_,image_,
type_,actionList_=[]):
 self.name=name_
 self.description=description_
 self.image=image_
 self.type=type_
 self.actionList_=actionList_
```

Vous noterez que je n'ai volontairement pas mis de propriété de prix, car celle-ci est une spécificité d'une boutique. On pourrait imaginer que certains personnages puissent offrir des objets, par exemple après que notre héros ait accompli une mission.

Ajoutons maintenant le code pour stocker ces objets dans l'inventaire de notre joueur, puis nous verrons la partie vente dans la suite de l'article.

Dans le fichier **common/singletons/GlobalPlayer.gd** :

```
...
var itemsList=[]
...
func addItem(item:item_class):
 itemsList.append(item)

func getItemsList():
 return itemsList
```

Nous pouvons ajouter également une classe pour les produits à vendre qui sera juste une enveloppe ajoutant le prix à ces objets, ce qui donnerait dans le fichier **common/class/shopItem_class.gd** :

```
extends Resource

class_name shopItem_class

var item : item_class
var price

func _init(item_,price_):
 self.item=item_
 self.price=price_
```

On aurait pu faire autrement en créant une classe **shopItem_class** qui hérite d'**item_class**, mais l'avantage avec cette sorte « d'enveloppe », c'est de simplifier la suite, à savoir l'ajout de ce simple objet dans l'inventaire du joueur, sans avoir à recréer à partir de **shopItem_class** un autre objet **item_class**.

2. GÉNÉRATION D'INTERFACE DYNAMIQUE

2.1 Listons des objets dans une boutique

Lorsque nous entrons dans une boutique, nous devons sélectionner parmi une liste d'objets disponibles. Nous pourrions créer un écran et créer à la main chaque élément à la vente avec ses propriétés, plus les boutons d'actions d'achat... ce qui serait chronophage. Nous préférerons ici gérer l'affichage ainsi que le bouton d'achat dynamiquement avec la puissance de Godot.

Créez un répertoire **ui** dans **common**, puis créez une nouvelle scène avec une racine **canvasLayer** et enregistrez-là dans **common/ui/shopList.tscn**.

Commençons par ajouter les éléments pour mettre en forme notre interface de sélection des produits dans la boutique. Je ne vais pas détailler de manière manuscrite chaque bouton à ajouter... que vous pourriez faire différemment selon vos goûts. Je vais plutôt détailler la partie qui nous intéresse : la conception dynamique d'interface.

En effet, l'idée est d'avoir un asset qui nous permettra facilement dans chaque boutique de proposer au joueur une sélection d'objets à acheter avec un prix défini.

Une fois l'interface créée, il nous faut deux choses importantes :

- un script sur le nœud racine qui permettra à l'extérieur, par exemple dans la scène de la boutique, de remplir la liste des objets à vendre ;
- un objet paramétrable qui sera dupliqué en boucle pour afficher les différents articles.

Pour rappel, l'ensemble du code de cet article est disponible sur le dépôt GitHub pour mieux voir en détail le code associé, notamment à cette partie.

L'interface terminée côté Godot ressemble à la figure 1.

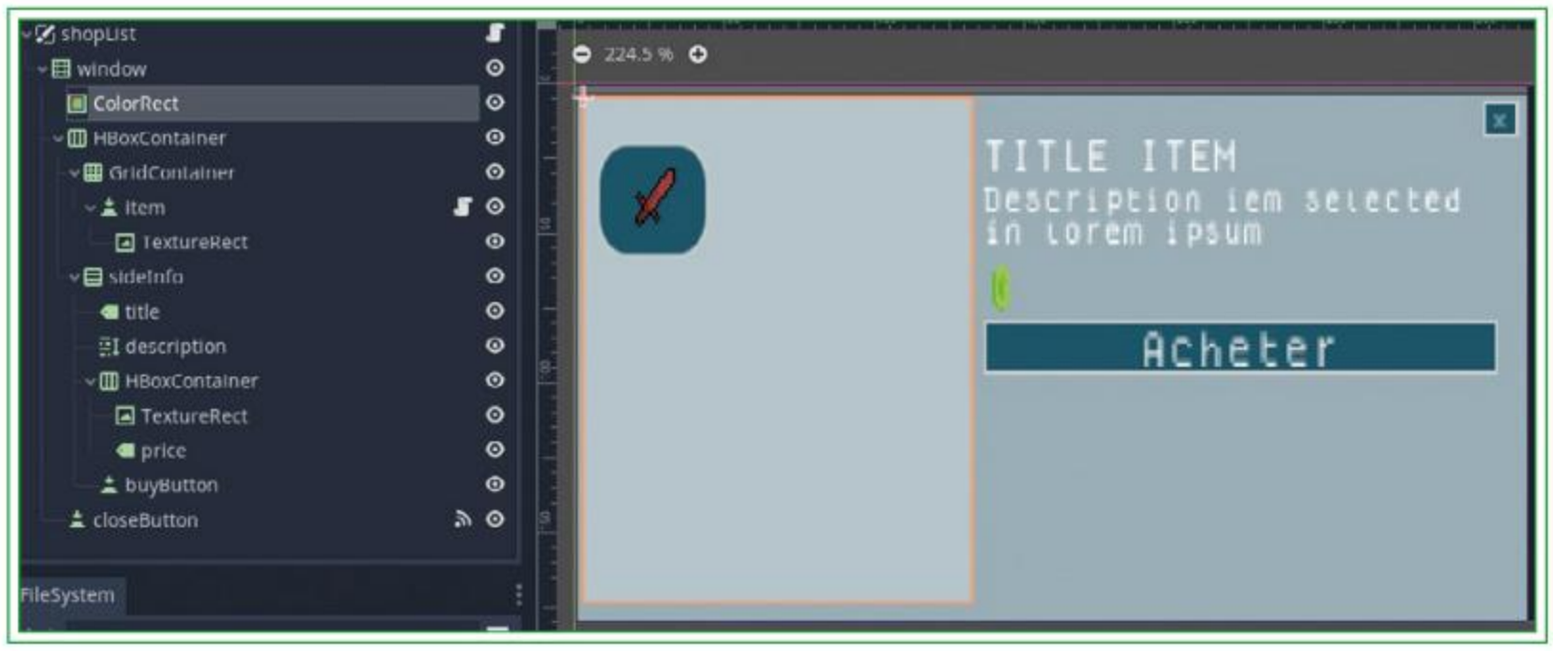


Fig. 1 : Scène de liste d'objets (pour les boutiques) côté Godot.

Dans un conteneur de type **GridContainer**, créons un simple bouton avec en enfant un **TextureRect** qui affichera l'icône de l'objet et enfin, ajoutez un script sur ce bouton qui permettra d'initialiser plus facilement l'objet complet.

Puis dans le script de cette scène, nous allons avoir ces parties intéressantes (fichier **common/ui/shopList.gd**) :

```
extends CanvasLayer

signal closePopup
signal buyItem(item_)

var shopItemList=[]
var itemSelected=null

var itemClass=preload("res://common/class/
item_class.gd")
var shopItemClass=preload("res://common/class/
shopItem_class.gd")
...
```

On charge nos deux classes d'objets, ainsi qu'un tableau pour les lister.

```
func _ready():
 getWindow().visible=false
 getSideInfo().visible=false
```

On cache au démarrage la fenêtre qui sera par défaut invisible dans la scène, jusqu'à ce que le scénario exige son affichage.

```
...
func show():
 getWindow().visible=true
 var exampleItem=getGrid().get_node("item")
 var patternItem=exampleItem.duplicate()
 exampleItem.queue_free()

 for shopItem in shopItemList:
  var newItem=patternItem.duplicate()
  newItem.setImage(shopItem.item.getTexture())
  newItem.connect("button_down",self,"_on_
pressed_selected",[shopItem])

  getGrid().add_child(newItem)
```

`show()` est une fonction pour charger de l'extérieur cette liste d'objet.

NOTE

Pour rappel, pensez à rendre générique ce genre de scène, plutôt que de multiplier à chaque boutique le même code.

Pour simplifier le processus, j'ai d'abord créé un seul objet dans cette grille avec une image pour deux raisons :

- voir le rendu attendu ;
- simplifier la partie dynamique par la suite.

Pour faire une partie dynamique, nous avons deux possibilités :

1. Soit créer tout dans le code : chaque élément, chaque paramétrage...
2. Soit créer graphiquement l'élément, puis dans le code le récupérer, pour servir de motif à répliquer.

C'est la deuxième solution que j'ai choisie ici : on commence donc par stocker l'objet exemple dans une variable, qui est utilisée juste après dans la boucle sur le tableau d'objets.

On définit en boucle l'image de l'icône, et on connecte le bouton à une méthode de sélection à laquelle on passe l'objet.

```
...
func _on_pressed_selected(shopItem_):
 getSideInfo().visible=true
 getSideInfo().get_node("title").
text=shopItem_.item.name
 getSideInfo().get_node("description").
text=shopItem_.item.description
 getSideInfo().get_node("HBoxContainer/
price").text=str(shopItem_.price)

 var buyButton=getBuyButton()
 buyButton.connect("button_down",self,"_
on_pressed_buy",[shopItem_])

 if GlobalPlayer.canSpendGems(shopItem_.
price):
  buyButton.disabled=false
 else:
  buyButton.disabled=true
```

L'objet sélectionné, on affiche dans la partie de droite son titre, sa description et son prix. En fonction de celui-ci, si le joueur peut l'acheter, on active ou non le bouton d'achat, bouton d'achat que l'on connecte lui aussi à une méthode pour acheter l'objet.

```
func _on_pressed_buy(shopItem_):
 GlobalPlayer.spendGems(shopItem_.price)
 GlobalPlayer.addItem(shopItem_.item)
 emit_signal("buyItem",shopItem_.item)
```

Sur cette méthode, on débite les « gems » côté joueur, on lui ajoute l'objet dans son inventaire et on émet le signal, par exemple si on veut ajouter un dialogue du vendeur pour lui confirmer son achat.

Pour tester cette scène, vous pouvez ajouter le code suivant dans la méthode de démarrage et lancer le rendu, vous aurez le rendu de la figure 2 (pensez à supprimer ce code avant la suite).

```
func _ready():
...
 setShopItemList([
  shopItemClass.new(itemClass.new("Epée
en bois","Une simple épée en bois de
hêtre","common/items/weapons/wood-sword.
png",itemClass.TYPE.WEAPON),10),
  shopItemClass.new(itemClass.new("Potion
de santé","Une potion de santé qui
donne 10 points de vie.","common/items/
magicpotion/magic-potion.png",itemClass.
TYPE.MAGICPOTION),50),

 ])
 show()
```

Fig. 2 : Le rendu de l'interface de saisie d'un objet dans une boutique.

2.2 Gestion de dialogues

Dans notre jeu d'aventure, nous allons communiquer avec certains personnages, pour éviter de réinventer la roue à chaque scène, nous allons de nouveau créer un asset réutilisable autant que nécessaire.

Créez une scène avec comme racine un **canvasLayer** et enregistrez-la dans **common/ui/simpleDialog.tscn**.

Ici encore, je ne vais pas détailler étape par étape la création de chaque nœud, je vais plutôt résumer la construction et insister sur les éléments intéressants.

Dans les grandes lignes, nous allons :

- afficher un objet texte pour le dialogue ;
- afficher un autre objet pour le nom du personnage prenant la parole ;
- nous afficherons le texte parlé caractère par caractère ;
- nous afficherons à chaque fin de phrase un bouton pour la suite ;
- en fin de dialogue, nous enverrons un signal pour indiquer à la scène la fin du dialogue.

Sur notre premier nœud, nous ajoutons un script **common/ui/simpleDialog.gd** que nous allons détailler :

```
extends CanvasLayer

signal discussionFinished

var multiDialogList=[]
var currentDiscussionPage=0
var currentDiscussionLine=0

func addDiscussion(talker_,discussionList_):
 multiDialogList.append({"talker":talker_,
"discussionLineList":discussionList_})

func start():
 getTalker().text=getCurrentDiscussionTalker()
 getDiscussion().
text=getCurrentDiscussionLine()
 getWindow().visible=true
 $Timer.start()

func end():
 getWindow().visible=false
 $Timer.stop()

func _ready():
 resetCharacterVisible()
```

Nous avons ici un signal de fin de discussion qui permettra à la scène d'indiquer quoi faire après, notamment fermer la fenêtre de discussion, mais nous verrons que l'on peut enchaîner sur un autre scénario.

Vous voyez également une méthode qui va nous permettre dans chaque scène de définir le dialogue en ajoutant via un objet quel personnage parle, ainsi qu'un tableau contenant ses lignes de texte.

Enfin, on démarre un chronomètre qui permettra l'affichage de chaque caractère petit à petit pour donner une animation de dialogue (et on rend visible la fenêtre de dialogue).

```
...
func displayNextCharacter():
 getDiscussion().set_visible_
characters(getDiscussion().get_visible_
characters()+1)
 if(getDiscussion().get_total_character_count() >
getDiscussion().get_visible_characters() ):
  getNextButton().visible=false
 else:
  getNextButton().visible=true

func _on_Timer_timeout():
 displayNextCharacter()

func _on_Button_pressed():
 next()
```

Régulièrement, on affiche un caractère de plus du dialogue et si la ligne est terminée, on affiche un bouton permettant de passer à la suite (pour laisser le temps de lire). Sur ce bouton, on va gérer le déroulement de ce dialogue.

```
...
func next():
 if getDiscussion().get_visible_characters() >
getDiscussion().get_total_character_count():
  if shouldContinueNextLine():
   currentDiscussionLine+=1
   resetCharacterVisible()
   getDiscussion().text=getCurrentDiscussionLine()
  elif shouldContinueNextPage():
   currentDiscussionPage+=1
   currentDiscussionLine=0
   resetCharacterVisible()
   getTalker().text=getCurrentDiscussionTalker()
   getDiscussion().text=getCurrentDiscussionLine()
  else:
   currentDiscussionPage=0
   currentDiscussionLine=0
   resetCharacterVisible()
   emit_signal("discussionFinished")
...
```

Premier cas, on n'a pas fini d'afficher le bloc de parole du personnage, on passe donc à la ligne suivante.

Second cas, on a fini la partie du personnage, et on passe au personnage suivant, changeant au passage le nom de l'orateur.

Enfin, on a fini l'échange, on peut tout remettre à zéro dans le cas où le joueur relancerait la discussion et on émet un signal à destination de la scène.

```
...
func getCurrentDiscussionLine():
 return getCurrentDiscussion()
["discussionLineList"][currentDiscussionLine]
...
```

J'affiche juste ici cette méthode pour montrer la manière d'afficher sur la même ligne l'accès à une propriété d'un objet au sein d'un tableau en GDScript.

NOTE

Étant adepte de la philosophie du « clean code » **[2]**, j'essaie au maximum de remplacer les conditions « techniques » par des méthodes parlantes, cela facilite la lecture de l'algorithme.

C'est pour cela que vous avez par exemple `shouldContinueNextLine()` plutôt que la condition vérifiant si le nombre de caractères est inférieur au nombre total.

2.3 Utilisation des objets dynamiques au sein d'une scène

Nous pouvons permettre l'achat, et dialoguer. Voyons comment utiliser ceux-ci au sein par exemple de notre boutique.

Ouvrez votre scène **screens/tree-village/shop.tscn**. Pour ajouter nos deux objets, vous devez simplement cliquer sur l'icône « lien » à droite du bouton **+** et sélectionner les objets **common/ui/shopList.tscn** et **common/ui/simpleDialog.tscn**. Nous allons maintenant les initialiser avant de les afficher. Cela se passe dans le fichier **screens/tree-village/shop.gd** :

```
var itemClass=preload("res://common/class/item_
class.gd")
var shopItemClass=preload("res://common/class/
shopItem_class.gd")


func _ready():
 $simpleDialog.addDiscussion("Vendeur",["Bonjour
Gordon","Que souhaites-tu ?"])
 $simpleDialog.addDiscussion("Gordon",["Bonjour
monsieur","Je souhaiterais acheter un objet"])
 $simpleDialog.addDiscussion("Vendeur",["Je vais
t'afficher les produits en vente","Tu n'auras
qu'à choisir"])
```

```
 $shopList.setShopItemList([
  shopItemClass.new(itemClass.new("Epée
en bois","Une simple épée en bois de
hêtre","common/items/weapons/wood-sword.
png",itemClass.TYPE.WEAPON),10),
  shopItemClass.new(itemClass.new("Potion
de santé","Une potion de santé qui
donne 10 points de vie.","common/items/
magicpotion/magic-potion.png",itemClass.
TYPE.MAGICPOTION),50),
 ])
```

Pour rappel, les objets sont cachés dans la scène par défaut, nous allons ajouter les événements pour les afficher. Ajoutons au départ l'asset **door** créé précédemment devant le comptoir du vendeur, renommons-le **discussionArea**, et ajoutons un événement pour afficher la discussion. Dans l'onglet ***node***, double cliquez sur le signal **playerOpenedDoor** :

```
func _on_discussionArea_playerOpenedDoor():
 $simpleDialog.start()
```

Nous démarrerons donc la séquence de dialogue en positionnant notre joueur devant le comptoir.

Dans l'onglet ***node*** de **simpleDialog**, double cliquez sur le signal **discussionFinished** :

```
func _on_simpleDialog_discussionFinished():
 $simpleDialog.end()
 $shopList.show()
```

Et en fin de discussion, nous cesserons le dialogue pour afficher la fenêtre de choix d'objets à acheter.

3. CRÉATION D'UNE INTERFACE TACTILE DE DÉPLACEMENT

3.1 Création de l'interface

Notre jeu est prévu pour être jouable sur écran tactile : smartphone et tablette, pour cela nous allons devoir modifier la manière dont nous déplaçons le héros à l'écran et ceci, en ajoutant un contrôleur tactile.

Au préalable, nous aurons besoin de deux cercles, un grand et l'autre deux fois plus petit : en effet, nous déplacerons avec notre pouce le petit cercle dans la zone du grand, comme on le ferait avec un joystick.

Créez une nouvelle scène avec un **canvasLayer** en racine, puis ajoutez les deux images dans la scène, les uns enfant de l'autre. Sur le plus grand des cercles, nous allons ajouter un script **common/ui/gamepad/navigation/joystick-big.gd** :

```
extends Sprite

signal moveJoystick(joystickVector)

const RADIUS = 32
const SMALL_RADIUS = 16

var joystick_pos;
var isPressed=false

#access
func getSmall():
 return $"joystick-small"

func _init():
 joystick_pos = position;

func _input(event_):
 if event_ is InputEventScreenTouch:
  if event_.is_pressed():
   if joystick_pos.distance_to(event_.position) <
RADIUS:
    isPressed=true

  else:
   isPressed=false
   getSmall().position=Vector2()
   emit_signal("moveJoystick",getSmall().position)

 if isPressed and event_ is InputEventScreenDrag:
  var joystickDistance= joystick_pos.distance_
to(event_.position)
  var joystickVector=(event_.position - joystick_
pos).normalized()

  if joystickDistance + SMALL_RADIUS > RADIUS:
   joystickDistance = RADIUS - SMALL_RADIUS

  getSmall().position=joystickVector*joystickDista
nce
  emit_signal("moveJoystick",getSmall().position)
```

En ajoutant une méthode **_input()**, on écoute tous les événements : on vérifie d'abord que c'est un événement tactile, que le doigt est pressé et que la position du doigt est dans la zone de notre grand cercle. Si c'est le cas, on peut stocker une variable pour indiquer que le doigt est appuyé (utilisé par la suite). Sinon, c'est que le doigt est relâché, dans ce cas, on repositionne notre petit cercle au centre.

Dans le cas où l'on sait le doigt pressé et qu'un événement de déplacement est détecté, on récupère la distance entre le point de départ et le doigt : le centre des deux cercles et la position du doigt, si celle-ci dépasse le grand cercle, on limite cette distance à celui-ci.

On déplace le petit cercle au produit de la différence entre la position d'origine et celui du doigt, vecteur multiplié par la distance calculée.

Enfin, on émet un signal pour indiquer cette nouvelle position.

Si vous testez sur votre PC, vous ne pourrez pas vérifier que cela fonctionne : cliquez sur ***project settings Input > Pointing devices*** et cochez ***Emulate touche from Mouse***. Ainsi, vous pourrez simuler à la souris l'appui du doigt.

3.2 Connexion de l'interface au joueur et déplacement

Dans notre scène de village ou de boutique, ajoutez via l'icône « lien » notre interface de navigation, puis cliquez sur l'onglet ***node*** pour connecter notre signal **movePlayer** à notre objet **player**. Il va falloir, comme d'habitude, créer une nouvelle méthode dans **common/characters/player.gd** que nous remplirons ainsi :

```
func _on_navigation_movePlayer(joystickVector_):
 useTouch=true

 right=false
 left=false
 down=false
 up=false

 if abs(joystickVector_.x)>abs(joystickVector_.y):

  if(joystickVector_.x > 10):
   right=true
 elif(joystickVector_.x < -10):
  left=true

else:

 if(joystickVector_.y > 10):
  down=true
 elif(joystickVector_.y < -10):
  up=true
```

Lors de la réception de nouvelles coordonnées, nous allons, après avoir réinitialisé les quatre directions, indiquer en fonction de l'intention absolue de déplacement la bonne direction. Notez que l'on ajoute une variable pour indiquer que l'on utilise le tactile, nous allons en effet permettre une gestion hybride du mode de déplacement : clavier et/ou tactile. Cette variable à ajouter en haut de votre script jouera un rôle surtout ici :

```
func resetKeys():
 if useTouch:
  return

 left=false
 right=false
 up=false
 down=false
```

Dans le cas de la réinitialisation des directions écrites précédemment, quand nous déplacions le personnage au clavier, on bypass cette partie dans le cas ou l'on utilise le joystick tactile.

D'ailleurs, c'est pour cela que nous la remettrons à faux dans le cas de l'utilisation du clavier.

```
func pocessInput():
 if Input.is_action_pressed("ui_right"):
  right=true
  useTouch=false
 if Input.is_action_pressed("ui_left"):
  left=true
  useTouch=false
 if Input.is_action_pressed("ui_down"):
  down=true
  useTouch=false
 if Input.is_action_pressed("ui_up"):
  up=true
  useTouch=false
```

Désormais, vous pouvez utiliser le clavier ou la souris (via le joystick) pour déplacer le personnage. Ce sera plus pratique pour tester le jeu sur votre ordinateur ou si vous souhaitez proposer les deux interfaces au joueur.

4. LE SYSTÈME DE COMBAT

4.1 L'interface de combat

Pour le système de combat, nous avons deux choix :

1. Soit utiliser le système de déplacement actuel en ajoutant des animations d'attaque et gérer les collisions avec les ennemis ;

2. Soit se tourner vers un mode de combat tour par tour. J'ai choisi la seconde option qui permet en plus de voir comment gérer un séquençage d'animation.

Nous allons nous concentrer ici sur quatre choses :

- les actions disponibles ;
- la gestion des attaques ;
- les séquences d'animations entre les attaques ;
- les dommages et la jauge de vie.

4.2 Les actions disponibles

Lors d'un combat, nous pouvons :

- soit nous enfuir et dans ce cas, on émet un simple signal à la scène pour nous faire retourner à la scène précédente ;
- soit attaquer, ceci dépendant bien sûr des armes à notre disposition.

Enfin, nous pouvons piocher dans notre inventaire, par exemple pour utiliser des potions de soin ou de poison qui pourraient nous aider à vaincre l'ennemi.

4.3 La gestion des attaques

Lors de l'initialisation de la boutique, nous avons créé une liste d'objets, dont une arme, mais nous n'avons pas rempli son tableau d'actions. Nous pouvons éditer le code précédent pour ajouter une liste d'attaques composée de plusieurs propriétés : son nom, son animation, sa puissance d'attaque et enfin le niveau d'expérience minimum requis. Ainsi, au fil de l'aventure, on pourra permettre à notre héros, avec la même arme, d'attaquer avec plus d'efficacité.

Nous allons d'abord créer deux classes pour stocker nos attaques et nos effets de potion (fichier **common/class/item/attack_class.gd**) :

```
extends Resource

class_name attack_class

var name
var animation
var damage
var xpMin
```

```
func _init(name_,animation_,damage_,xpMin_):
 self.name=name_
 self.animation=animation_
 self.damage=damage_
 self.xpMin=xpMin_
```

Et dans le fichier **common/class/item/potion_class.gd** :

```
extends Resource

class_name potion_class

enum ACTION {INCREASE_LIFE,INCREASE_MAGIC}

var action
var value

func _init(action_,value_):
 action=action_
 value=value_
```

Avec ces objets définis, nous pouvons ajouter pour chaque arme une liste d'attaques.

Éditez le fichier **screens/tree-village/shop.gd** :

```
func _ready():
 ...
 $shopList.setShopItemList([
  shopItemClass.new(itemClass.new(
   "Epée en bois",
   "Une simple épée en bois de hêtre",
   "common/items/weapons/wood-sword.png",
   itemClass.TYPE.WEAPON,
   [
    attackClass.new(
     "Attaque simple",
     "attack_wood-sword_simple",
     10,
     0
    ),
    attackClass.new(
     "Attaque avancée",
     "attack_wood-sword_advanced",
     20,
     10
    ),
    attackClass.new(
     "Attaque en feu",
     "attack_wood-sword_onfire",
     50,
     50
    )
   ]),10),
```

```
  shopItemClass.new(itemClass.new(
    "Potion de santé",
    "Une potion de santé qui donne 10 points
de vie.",
    "common/items/magicpotion/magic-potion.
png",
    itemClass.typeList.MAGICPOTION,
    [
     potionClass.new("increaseLife",10)
    ]
    ),50),
])
```

Notez que l'on a défini pour notre arme une série d'attaques et pour notre potion, une action à effectuer. On pourrait imaginer des potions de vie, de pouvoir de magie, de force, etc. selon les jeux.

Maintenant, nous devons ajouter à notre classe joueur une méthode pour retourner les attaques disponibles en fonction des armes qu'il possède.

Dans le fichier **common/singletons/globalPlayer.gd** :

```
extends Node

var position=null
var gems=0
var itemsList=[]
var xp=0
```

Après avoir ajouté notre expérience, qui sera incrémentée à chaque combat victorieux, nous ajoutons une méthode pour nous retourner notre tableau d'attaques disponibles :

```
...
func getAttackList():
 var attackFilteredList=[]
 for item in getItemsList():
  if item.type==item_class.TYPE.WEAPON:
   for attack in item.actionList:
    if xp >= attack.xpMin:
     attackFilteredList.append(attack)

 return attackFilteredList
```

Dans notre scène d'attaque, nous pourrons ainsi au moment du clic sur le bouton d'attaque lister celles-ci.

Dans **common/gamePad/fight.gd** :

```
...
func build():
 for attack in GlobalPlayer.getAttackList():
  var buttonTmp=Button.new()
  buttonTmp.text=attack.name

  buttonTmp.connect("button_
down",self,"_on_attack_
pressed",[attack])

  $attackList.add_child(buttonTmp)
```

Qui émettra pour la scène de combat un signal de l'attaque à lancer :

```
func _on_attack_pressed(attack_):
 emit_signal("attack",attack_)
```

4.4 Les personnages et leurs animations

Il nous reste encore une étape : créer les différentes animations. Avec votre logiciel habituel, créez vos sprites d'animation. Créez ensuite une scène pour chacun des personnages, composée d'un **animatedSprite** et d'un script.

Dans l'animation, ajoutez autant d'animations que votre personnage peut posséder d'attaques, ainsi qu'une animation de dommages qui sera jouée lors d'une attaque subie.

Enfin, ajoutez dans le script le code qui permettra de jouer les animations d'attaque et de dommages, ainsi que les différents signaux pour permettre leur séquencement.

Dans le fichier **common/fighters/player.gd** :

```
extends Node2D

signal attackFinished
signal damageFinished
signal dieFinished

var currentAnim=TYPE_ANIM.IDLE

enum TYPE_ANIM {ATTACK,DAMAGE,IDLE,DIE}

func playAttack(animation_):
 $AnimatedSprite.play(animation_)
 currentAnim=TYPE_ANIM.ATTACK

func playDamage():
 $AnimatedSprite.play("damage")
```

```
 currentAnim=TYPE_ANIM.DAMAGE

func playIdle():
 $AnimatedSprite.play("idle")
 currentAnim=TYPE_ANIM.IDLE

func playDie():
 $AnimatedSprite.play("die")
 currentAnim=TYPE_ANIM.DIE

func stop():
 $AnimatedSprite.stop()

func _on_AnimatedSprite_animation_finished():
 stop()
 if currentAnim==TYPE_ANIM.ATTACK:
  emit_signal("attackFinished")
 elif currentAnim==TYPE_ANIM.DAMAGE:
  emit_signal("damageFinished")
 elif currentAnim==TYPE_ANIM.DIE:
  emit_signal("dieFinished")
 elif currentAnim==TYPE_ANIM.IDLE:
  playIdle()
```

Notez que l'on s'arrête à chaque fin d'animation pour donner la main à la scène, sauf pour le cas de l'**idle** qui est l'animation d'attente à jouer en boucle.

Pour information, le joueur et notre premier ennemi auront le même script.

4.5 La jauge de vie

Nos deux personnages ont une jauge de vie qui devra évoluer au fil des attaques : pour cela, vous pourrez à nouveau créer un **asset** en utilisant cette fois l'objet **progressBar** et en ajoutant un script pour permettre d'agir dessus.

Voici le code pour initialiser le nombre de vies ainsi que sa jauge maximum : en effet, certains ennemis auront plus ou moins de vie que notre héros (fichier **common/ui/lifeBar.gd**) :

```
extends Control

func init(currentValue_,maxValue_):
 $ProgressBar.value=currentValue_
 $ProgressBar.max_value=maxValue_

func updateValue(value_):
 $progressBar.value=value_
```

4.6 La cinématique des animations

Maintenant, nous allons dans la scène de combat mettre ceci en musique.

Après avoir créé une scène, nous allons devoir ajouter nos deux protagonistes sur celle-ci et ajouter un script pour tout orchestrer.

Créez une scène dans **screens/macro-map/spiderFight.tscn** avec bien sûr un script également, et renommez nos deux personnages **player** et **spider1**. Puis connectez les signaux sur notre script. Vous devriez avoir le code suivant (fichier **screens/macro-map/spiderFight.tscn**) :

```
extends Node2D

var lifeSpider=50
var damageSpider=10


func _ready():
 $lifePlayer.init(GlobalPlayer.
getLife(),GlobalPlayer.
getMaxLife())
 $lifeSpider.
init(lifeSpider,lifeSpider)
```

Au départ, on initie juste les barres de vies de nos deux personnages.

```
...
func _on_gamepad_
attack(attack_):
 $player.playAttack(attack_.
animation)
 lifeSpider-=attack_.damage

 $gamepad.hide()

func _on_player_
attackFinished():
 $spider1.playDamage()
```

On intercepte l'événement d'attaque de notre héros pour jouer l'animation et décrémenter les vies de l'adversaire

en fonction de l'attaque utilisée, on désactive la possibilité de mouvement pour éviter un doublon d'attaque. On attend la fin de l'animation d'attaque pour jouer celle des dommages, côté ennemi.

```
func _on_spider1_damageFinished():
 $lifeSpider.updateValue(lifeSpider)
 if lifeSpider <=0:
  $spider1.playDie()
 else:
  $spider1.playAttack("attack-1")
  GlobalPlayer.damage(damageSpider)
```

À la fin de l'attaque, après avoir mis à jour la jauge de vie, on donne la main à notre ennemi pour jouer son attaque. Ici, une seule attaque, mais on pourrait imaginer notamment pour des « boss » des attaques au hasard ou par séquence.

```
func _on_spider1_attackFinished():
 $player.playDamage()

func _on_player_damageFinished():
 $lifePlayer.updateValue(GlobalPlayer.getLife())

 $gamepad.show()

func _on_spider1_dieFinished():
 GlobalPlayer.addXp(1)
 get_tree().change_scene("res://screens/macro-map.tscn")

func _on_player_dieFinished():
 get_tree().change_scene("res://screens/game-over.tscn")
```

On fait la même chose en miroir côté joueur.

Et le dernier événement permet d'ajouter de l'expérience, puis de quitter la scène de combat en cas de victoire ou de charger l'écran de fin de partie, dans le cas contraire. Ce qui nous donne le résultat visible sur la figure 3.

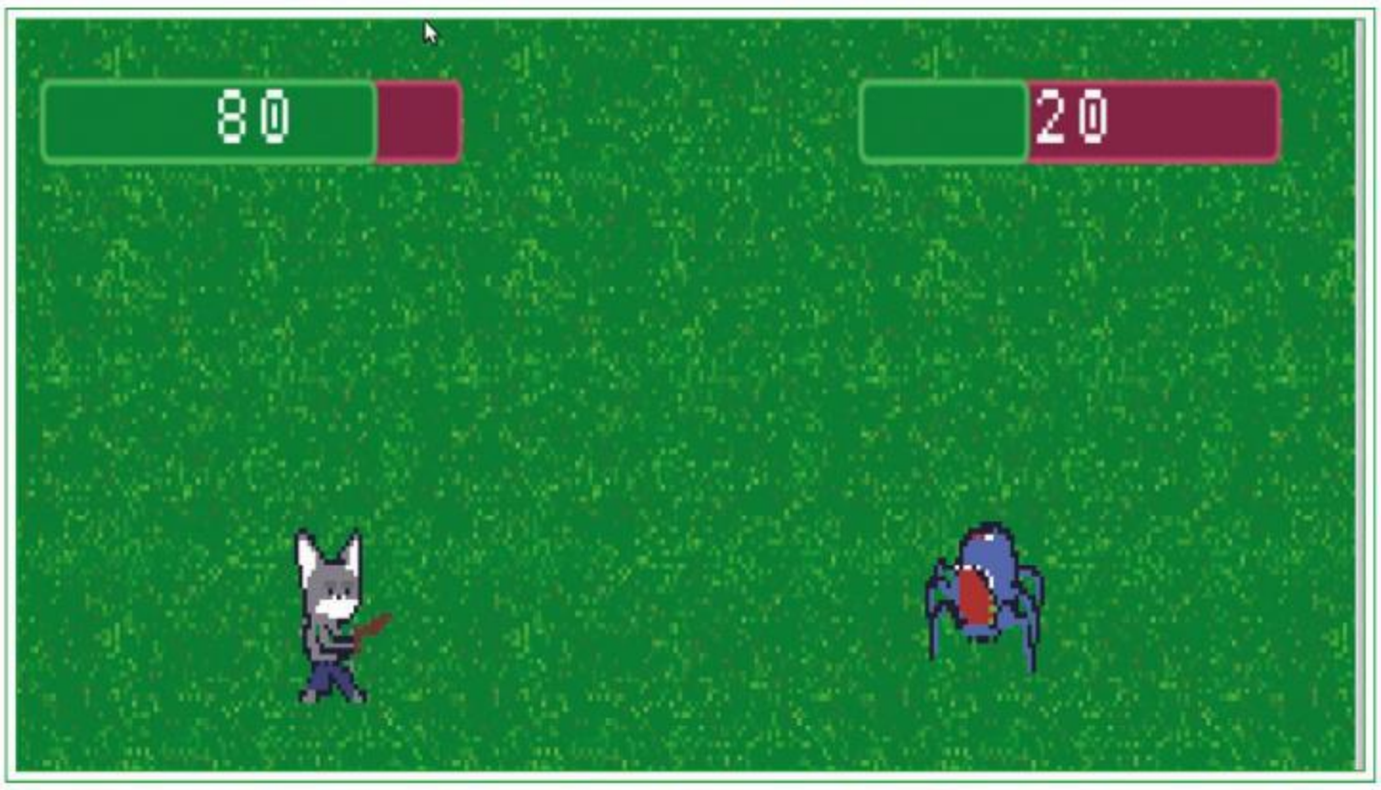


Fig. 3 : Scène de combat type tour par tour.

CONCLUSION

Nous avons pu voir ici les grandes lignes des petits « défis » techniques à relever pour ce type de jeu.

Vous avez pu voir qu'il faut au maximum penser vos développements génériques et paramétrables, et vous avez pu également voir que Godot vous permet de facilement générer des éléments d'interface.

Nous avons vu également comment créer et utiliser des classes d'objets pour gérer des objets plus complexes tels que des armes, qui une fois possédées permettent lors des phases de combat de proposer différentes attaques. Vous avez vu comment on peut également créer un séquencement d'animations.

Nous approchons de la fin de cette série d'articles : il nous reste encore l'utilisation des tests unitaires, la sauvegarde/le chargement de partie, la mise en place d'un menu de jeu, et les différents types d'export web et smartphone... ■

RÉFÉRENCES

[1] M. BERTOCCHI, « Godot : Comment créer un jeu d'aventure », GNU/Linux Magazine n°247, avril 2021 : https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-247/Godot-comment-creer-un-jeu-d-aventure

[2] R. MARTIN, « Clean code : A Handbook of Agile Software Craftsmanship », Pearson, 2008.

PARC EXPO DES OUDAIRIES
LA ROCHE-SUR-YON

DU 8 AU 10
JUILLET 2021

COUPE DE FRANCE DE ROBOTIQUE

Les robots prennent le large...
Suivez leur course !

INÉDIT
Finale de la Coupe Junior !

WWW.COUPEDEROBOTIQUE.FR

#CDR2021

EUROPEAN CYBER CUP

LA 1ÈRE COMPÉTITION DE ESPORT DÉDIÉE AU HACKING ÉTHIQUE

LE 8 & 9
SEPTEMBRE 2021
À LILLE
GRAND PALAIS

TO BE [PWNED]
OR NOT TO BE [PWNED]
THAT IS THE {CYBER}QUESTION

WWW.EUROPEAN-CYBERCUP.COM

Organisé par :

FIC 2021

En partenariat avec :

CYBERSÉCURITÉ OFFENSIVE & DÉFENSIVE
MISC
PAPIER + BASE DOCUMENTAIRE ONLINE

Avec le soutien de :

MEL
MÉTROPOLE
EUROPÉENNE DE LILLE