[sfeir]

Innovation et expertise
en développement
Back-End

SAVOIR SFEIR BACK

Par :

Yves DAUTREMAY, Développeur Agile et Back-End, SFEIR Strasbourg - **Fabian PIJCKE**, Développeur Agile et Back-End, SFEIR Luxembourg - **Guillaume FOURNY**, Développer Back-End Senior, SFEIR Nantes - **Alexandre MOEVI**, Développeur Back-End, SFEIR Lille - **Mathieu VANDENNEUCKER**, Développeur Full-Stack, SFEIR Belgium - **Gaëlle CHALOT**, Développeuse Full-Stack, SFEIR Strasbourg - **Erwan Le Tutour**, Ingénieur d'étude et de développement, SFEIR Paris - **Charlotte RYSSEN**, Content Manager et Rédactrice en Chef de sfeir.dev, SFEIR Groupe.

Sommaire

Avant-propos

Avec une expérience éprouvée et une approche centrée sur l'innovation, SFEIR se distingue comme un partenaire de choix pour les entreprises cherchant à transformer leurs systèmes d'information et à adopter les dernières avancées technologiques.

Excellence technologique

L'expertise de SFEIR dans le développement Back-End est ancrée dans une connaissance approfondie des technologies les plus récentes et les plus efficaces. Des langages de programmation tels que Java, Kotlin ou NodeJS jusqu'aux frameworks avancés comme Spring Boot et Quarkus l'équipe SFEIR maîtrise un éventail de technologies pour fournir des solutions robustes et performantes. Cette maîtrise technique permet à SFEIR de développer des systèmes Back-End qui ne sont pas seulement fonctionnels mais aussi optimisés pour l'évolutivité, la sécurité et la maintenabilité.

Innovation et R&D

La passion pour l'innovation est au cœur de l'identité de SFEIR. En investissant continuellement dans la recherche et le développement, SFEIR reste à la pointe des tendances technologiques, se positionnant ainsi comme un leader dans l'exploration de nouvelles solutions Back-End et de méthodologies de développement avancées. Que ce soit à travers l'adoption précoce de technologies émergentes ou le développement de solutions propriétaires, SFEIR démontre un engagement envers l'innovation qui bénéficie directement à ses clients.

Approche agile et collaborative

La méthodologie de travail de SFEIR est profondément ancrée dans les principes agiles. Cette approche garantit une grande réactivité et flexibilité, permettant des cycles de développement rapides et efficaces tout en maintenant une communication de proximité avec les clients. En collaborant étroitement avec ses partenaires, SFEIR assure que les solutions développées répondent précisément aux besoins spécifiques de chaque projet.

Engagement envers la sécurité et la qualité

La sécurité et la qualité sont des priorités absolues dans tous les projets de développement chez SFEIR. L'entreprise adopte les meilleures pratiques de l'industrie pour garantir que les applications sont non seulement performantes et évolutives, mais aussi sécurisées contre les menaces modernes. Cette attention à la sécurité est complétée par un engagement envers la qualité du code, assurant ainsi la livraison de solutions fiables et durables.

Un partenaire de confiance pour la transformation numérique

SFEIR se positionne comme un partenaire stratégique pour les entreprises en quête de transformation numérique. En alliant expertise technique, innovation continue et méthodes de travail agiles, SFEIR aide ses clients à naviguer dans le paysage technologique en constante évolution, en leur fournissant les outils et les compétences nécessaires pour réussir dans l'ère numérique.

Combinant une expertise technique approfondie, une passion pour l'innovation, et une méthodologie agile pour offrir des solutions de pointe qui propulsent les entreprises vers le succès numérique, SFEIR souhaite mettre en avant les membres et experts de sa communauté avec ce recueil d'articles.

Technologies avancées et tendances

Un panorama des solutions innovantes et des paradigmes émergents qui redéfiniront les standards de l'industrie.

À travers ces articles d'experts, cette section résume les tendances actuelles en matière de technologies Back-End, soulignant comment elles façonnent l'avenir du développement et la manière dont elles peuvent être intégrées dans les pratiques actuelles.

- **Quoi de neuf dans l'API Java 21 ?**

Cet article explore les dernières mises à jour de l'API Java 21, en mettant l'accent sur les nouvelles fonctionnalités et améliorations qui renforcent la performance et la facilité d'utilisation pour les développeurs.

- **Révolutionnez votre programmation asynchrone avec Java 21 et Loom**

Cet essai se concentre sur les avantages de Java 21 et Loom pour la programmation asynchrone, offrant des exemples pratiques pour illustrer leur utilité dans des scénarios réels.

- **Du bruit avec Kotlin**

Le potentiel de Kotlin en tant que langage de programmation moderne et polyvalent est discuté ici. L'article souligne comment Kotlin offre une alternative efficace à Java, en particulier pour le développement Android.

Quoi de neuf dans l'API Java 21?

Par Yves DAUTREMAY.

Java 21 est sorti en septembre 2023 et embarque une quinzaine de JEP. Voici un tour d'horizon de ses nouveautés.

Java 21 est sorti ! Outre le fait qu'il s'agisse d'une version LTS (Long Term Support), celle-ci est riche en innovations. Difficile de vous y retrouver ? Examinons cela en détail.

Nouveautés de l'API

Ces fonctionnalités sont d'ores et déjà opérationnelles dans Java 21.

Sequenced Collections (JEP 431)

Enfin, un renouvellement de l'API Collection ! Il manquait effectivement la notion de collections ordonnées. Trois nouvelles interfaces sont introduites dans cette mouture :

- SequencedCollection
- SequencedSet
- SequencedMap

Elles proposent des méthodes communes et cohérentes pour manipuler ou ajouter les éléments en début ou en fin de liste.

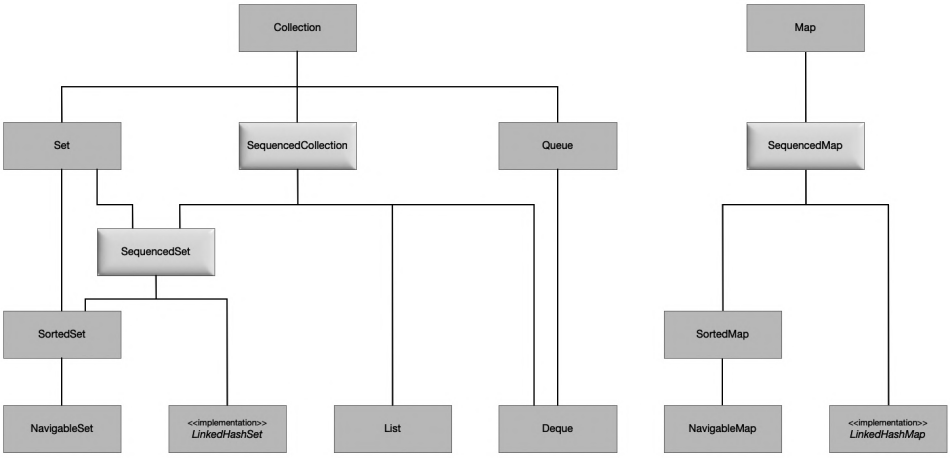


Diagramme de classes Collection API

Notons que l'objet ArrayList, qui implémentait déjà List, est dorénavant également une SequencedCollection.

Record Pattern (JEP 440)

Précédemment en phase de preview, cette fonctionnalité est désormais officielle et autorise la déconstruction d'un record pour accéder à ses sous-composants. En Java 16, le Pattern Matching nous permettait déjà de simplifier l'écriture d'un instanceof en évitant le cast.

```
if(o instanceof String){
    String s = (String)o;
    System.out.println(s);
}
```

Devenait alors :

```
if(o instanceof String s){
    System.out.println(s);
}
```

On va utiliser la structure des Records pour les déconstruire et accéder aux champs directement :

```
record Point(int x, int y) {}

if(o instanceof Point(int a, int b)){
    int c = a + b;
}
```

Ca marche aussi dans un switch

```
int c = switch (o){
    case Point(int a, int b) -> a + b
    ...
}
```

Threads Virtuels (JEP 444)

C'est LA nouveauté attendue par tous, qui sort enfin de sa phase de preview! Pour en savoir plus, je vous renvoie à mon article suivant. (Révolutionnez votre programmation asynchrone avec Java 21 et Loom).

Autres ajouts

Mentionnons quelques autres petites nouveautés intéressantes :

- Gestion des Emoji : *Character.isEmoji()* entre autres
- *StringBuilder.repeat()* pour répéter une String
- *HttpClient* implémente *AutoClosable* et peut être utiliser dans un bloc *try with resource*

Nouveautés en Preview

Ces fonctionnalités ne sont pas activées par défaut et nécessitent l'ajout de flags lors de la compilation.

Pour les activer il faut ajouter les flags

```
--release 21 --enable-preview
```

lors de la compilation, et

```
--enable-preview
```

lors du run.

String Templates (JEP 430)

Permet d'intégrer des expressions qui seront interpolées dans les chaînes de caractères. Si beaucoup de langages supportent déjà cette fonctionalité, Java permet de le rendre safe grâce à des étapes de validation nettoyage via un *Template Processor*. On peut ainsi se protéger des injection SQL par exemple.

Un *Template Processor* prend un template, et l'interpole vers un autre objet. On peut ainsi interpoler vers une *String, unPreparedStatement*, un *JSONObject*...

```
String str = "World"
String result = STR."Hello \{str}";
```

On utilise une expression avec la notation \ *{expression}* et on appelle un Processor avec sont nom, ici *STR*, qui est une constante de la classe StringTemplate, qu'on importe de façon statique.

Le JDK apporte 3 Processors :

- RAW : n'interpole pas les expression
- STR : interpole une String vers une String par concaténation
- FMT : interpole une String vers une String en autorisant le formattage via un *Formatter*

```
String log = FMT."%d\{a} + %d\{b} = %d\{a + b}
```

Il est possible de créer son propre Processor en implémentant l'interface *StringTemplate. Processor*

Variables et Patterns anonymes (JEP 443)

Cela arrive régulièrement de devoir déclarer une variable alors qu'elle est inutile. Dans le cas du catch d'exception ou d'une lambda par exemple. Cette nouveauté permet de ne pas avoir à la nommer en utilisant le caractère '_' (underscore).

```
//Variable locale d'un statement
for(Element _ : elements){
    //Bloc qui n'utilise pas les élément
}
var _ = mySet.remove(myObject);

//Bloc catch
try{
    int i = Integer.parseInt(str);
}catch(NumberFormatException _){
    logger.warn("Not a number");
}

//Lambda
Map<EmployeeId, List<Employee>> employees = new HashMap<>();
employees.computeIfAbsent(id, _ -> new ArrayList<>());
```

On peut aussi l'utiliser dans les Pattern Matching

```
if(object instanceof Point(int _, int y)){
    ...
}

int n = switch(obj){
    case Point(int x, int _) -> x
}
```

Unnamed Classes et Instance Main method (JEP 445)

Le but de cette JEP est de faciliter l'apprentissage de Java et de rendre optionnel tout ce qui nécessite de créer une méthode main, pour n'avoir à la fin que la méthode la plus simple possible. Par exemple, aujourd'hui, on écrit :

Java 21 : changements majeurs dans l'évolution du langage et améliorations significatives de la performance, de la sécurité et de l'expérience de développement.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Pour écrire Hello World dans la console, il faut décrire une grande quantité de concepts : classe, visibilité, static, la signature spécifique de main avec son tableau de String.

On simplifie d'abord en enlevant la visibilité, le static et les paramètres :

```
class HelloWorld {
    void main() {
        System.out.println("Hello World!");
    }
}
```

Enfin, on simplifie encore plus avec la notion de classe anonyme :

```
    void main() {
        System.out.println("Hello World!");
    }
```

Une classe anonyme est un fichier .class qui n'a pas de déclaration de classe, ne peut pas être appelé par une autre classe, mais peut contenir des méthodes et des champs.

Cela permet de faciliter l'écriture de la méthode main, et est surtout tournée pour un apprentissage du langage.

Liste officielle

Il y a encore d'autres nouveautés dans cette version, retrouvez la *release note* complète de Java 21. Finalement, la version 21 de Java apporte un éventail de nouveautés intéressantes, desquelles on pourrait passer à côté si l'on se focalisait exclusivement sur les threads virtuels. De surcroît, il s'agit d'une version LTS : vous n'avez donc aucune excuse pour ne pas l'employer dans vos nouveaux projets.

Java 21 :
un tournant majeur
dans l'ère du codage !

Révolutionnez votre programmation asynchrone avec Java 21 et Loom

Par Yves DAUTREMAY.

Depuis sa version 19, Java a introduit le projet Loom, et avec lui, la promesse d'une programmation asynchrone non bloquante, tout en restant dans un paradigme impératif.

Quel problème essaie-t-on de résoudre ?

Tout programme qui doit s'exécuter est associé à un Thread du Système d'Exploitation (OS). C'est ce dernier qui va décider de son exécution sur le CPU.

Dans l'absolu, on ne peut exécuter qu'un seul thread à la fois sur un CPU, le parallélisme étant simulé par l'OS en faisant tourner les Threads qui s'exécutent. Depuis l'arrivée des processeurs multicœur, on peut exécuter réellement en parallèle autant de Thread que de cœur de processeur.

• En Java, pour exécuter du code, on utilise la classe **Thread,** introduite en 1995, qui représente précisément un seul Thread système. On peut exécuter une portion de code sur un Thread en lui passant un Runnable (dans l'exemple ci-dessous, une lambda qui remplace l'interface fonctionnelle Runnable).

```
public class Main {
    public static void main(String[] args) throws Exception {
        Thread.ofPlatform()
                .start(() -> System.out.println("In thread"))
                .join();
    }
}
```

Il est important de noter qu'un Thread OS est relativement coûteux à créer et que, en général, les applications vont créer un Pool de Thread pour optimiser l'exécution.

• Tout cela fonctionne très bien, tant qu'on a une exécution continue. Dans l'idéal, on voudrait que le CPU soit utilisé au maximum de ses capacités. Dans la réalité, les Threads sont souvent bloqués par de l'attente IO. Typiquement, lors d'une requête à une base de données, le Thread va attendre plusieurs millisecondes la réponse avant d'avoir le retour pendant lequel il ne peut rien faire. Avec un peu de chance, l'OS passera à l'exécution d'un autre Thread mais, dans une application de gestion, il y a une forte probabilité que la plupart des Threads en cours soient pendus à l'attente d'une réponse. Une fois le pool de threads épuisé, **l'application est paralysée.**

Loom et les Threads Virtuels

Contrairement au Thread Plateforme qui est associé à un unique Thread de l'OS, qui sera d'ailleurs créé pour l'occasion, un Thread Virtuel n'est associé à aucun Thread. Au moment de son exécution, la JVM va choisir un Thread porteur (Carrier Thread) et y affecter le Runnable. Lorsque le code fait une opération bloquante, il va être détaché du Thread porteur (Yield) et son état va être stocké en RAM. Le thread porteur est alors libre d'exécuter le code d'un autre Thread virtuel (via une Continuation).

• La plupart des opérations bloquantes de la JVM (sleep, opérations sur fichiers, ...) y compris le pilote JDBC, sont mises à jour pour tirer parti de ce mécanisme. On peut donc s'attendre à de grandes améliorations des performances, **sans avoir à changer le code que l'on a l'habitude d'écrire.**

Voici un petit exemple des capacités de Loom pour gérer le code bloquant

```
public class Main {
    public static void main(String[] args) throws Exception {
        LocalDateTime time = LocalDateTime.now();
        Runnable r = () -> {
            try{
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        };
        List<Future<?>> futures = new ArrayList<>(100_000);
        ExecutorService virtualThreadExecutor = Executors.newVirtualThreadPerTaskExecutor();
        for (int i = 0; i < 100_000; i++){
            futures.add(virtualThreadExecutor.submit(r));
        }
        for(Future<?> f : futures){
            f.get();
        }
        System.out.println(ChronoUnit.MILLIS.between(time, LocalDateTime.now()));
    }
}
```

On déclare un Runnable qui va dormir 5 secondes. On va lancer ce Runnable 100.000 fois dans un ExecutorService qui lance un Thread Virtuel pour chaque nouvelle tâche. On attend ensuite la fin des threads via Future.get().

Sortie console :

```
6319

Process finished with exit code 0
```

Soit 6,3 secondes.

Je déconseille fortement de lancer tout ça dans un Executor classique. Pour espérer avoir une exécution dans les 5 secondes il faudrait créer un pool de 100.000 Threads, ce qui prendra bien plus que 5 secondes et sera très couteux en ressources système.

Limitations

- L'utilisation des blocs de code **synchronized** contenant du code bloquant ne sera pas géré par Loom. Il faut éviter leur utilisation et préférer un **ReentrantLock.** Malheureusement, l'effet est aussi présent pour une librairie qui utiliserait un bloc **synchronized.**
- L'utilisation de Threads Virtuels n'est pas recommandée si une exécution ne fait pas d'opération bloquante et fait surtout du calcul. On s'évite l'overhead de la gestion des Carriers.

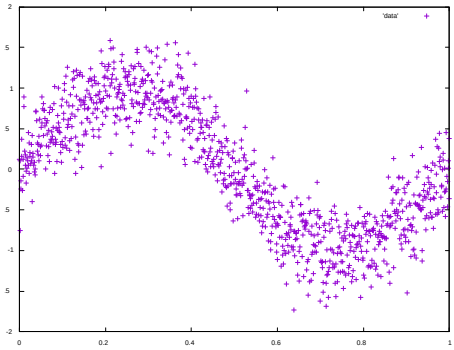
Conclusion

L'utilisation de Loom et des Threads Virtuels va devenir fondamentale dans toutes les applications web/métiers qui font beaucoup d'opérations bloquantes (requêtes BDD, HTTP, ...). Des gains de performances importants seront constatés sans même avoir à changer le code.

L'équipe Spring Boot travaille sur son intégration depuis la préversion de Java 19. Il ne fait aucun doute qu'une version incluant ces fonctionnalités sera publiée peu de temps après la sortie de Java 21.

Du bruit avec Kotlin

Par Fabian PIJCKE.



Les IA sont à la mode ! Pour entrainer une IA, il faut de bons jeux de données. Pour faire de bons jeux de données, il faut du code, et je vous propose ici de voir ensemble une version de départ d'un générateur de données qui suit une tendance tout en ajoutant juste ce qu'il faut d'aléatoire.

Introduction

Avec ChatGPT qui prend le monde d'assaut en ce moment, je n'ai pas pu m'empêcher de me remettre à la lecture de l'excellent livre Pattern Recognition and Machine Learning par Christopher M. Bishop (en fait, je dis excellent mais je n'en sais rien, je ne suis pas encore allé plus loin que le chapitre 2, mais jusque là j'adore ce bouquin, et il y a plein d'exercices ! :-))

C'est le genre de bouquin qu'on lit avec un PC devant soi pour rendre le texte plus compréhensif en faisant plein d'expériences. Mon langage préféré du moment étant Kotlin, et le code étant assez sympa pour un premier contact avec ce langage, je vous fais profiter d'un mini projet qui génère des points en vue d'être utilisés par un algorithme de régression.

Voilà le topo : on veut entraîner un modèle qui va approximer une courbe par un polynome. Cette courbe est générée à partir de la fonction sinus sur une seule période avec un bruit gaussien dont la déviation standard est 0.3. Cette valeur est donnée dans l'annexe A du bouquin pour ceux qui vérifient que je ne dis pas n'importe quoi. Le fact checking va probablement devenir encore plus important dans les prochains mois/années mais je pense qu'il y a quand même mieux à faire que fact-checker un article technique comme celui-ci ! Enfin c'est votre vie :-)

Quelques mots sur Kotlin

Kotlin est un langage dérivé de Java qui tourne sur la JVM. L'un des avantages de Kotlin est son interopérabilité avec Java : Vous pouvez ajouter des classes Kotlin dans un projet Java, ajouter le bon plugin dans Maven/Gradle, et tout tourne ! Vos classes Java peuvent utiliser la classe Kotlin que vous venez d'introduire et inversement.

Kotlin fait la part belle au paradigme fonctionnel, même s'il reste principalement orienté objet.

Un autre point fort de Kotlin est le fait qu'il soit bien moins verbeux que Java (même si Java fait des efforts en ce sens depuis plusieurs années). Il semble d'ailleurs y avoir une bonne synergie entre Java et Kotlin puisque les deux projets s'inspirent mutuellement.

Générer des points suivant une fonction

Les points à générer doivent répondre à plusieurs exigences : Suivre la courbe du sinus, et avoir un bruit aléatoire. Je suis un grand fan du travail de l'oncle Bob et je pense qu'il s'agit là de deux responsabilités distinctes qui doivent donc être gérées par deux classes distinctes. J'ai dans l'idée d'utiliser un décorateur pour ajouter le bruit mais on en parlera plus tard. Pour le moment, contentons-nous d'écrire le code pour générer les points selon une fonction donnée.

Ah on avait dit sinus ? Bon, un peu de généralisation prématurée n'a jamais fait de mal à personne (et non, ça ne remet pas en cause mon respect pour le travail de l'oncle Bob). Allez hop ! Voici comment on écrit une interface en Kotlin :-)

```
interface Generator {
    fun generate(x : Double) : Double
}
```

Vous connaissez les implémentations par défaut dans les interfaces en Java ? Eh bien en Kotlin vous pouvez simplement fournir des implémentations dans les interfaces, tant que l'interface n'a pas de champ, vous êtes tout bon.

Profitons-en donc pour ajouter une fonction qui génère un nombre donné équiréparti de points entre deux bornes :

```
interface Generator {

    fun generate(x : Double) : Double

    fun generatePoints(xMin : Double, xMax : Double, count : Int) :
        return List(count) {
            val x = xMin + it * (xMax - xMin) / (count - 1)
            Point(x, generate(x))
        }
    }

}
```

Comme vous le constatez, pas de point-virgule en fin de ligne, pas besoin de déclarer le nom du paramètre dans une lambda (il est nommé `it` par défaut), et les bindings sont introduits par `val`. On peut aussi utiliser `var` si la valeur est amenée à changer, mais Kotlin nous encourage à déterminer si nous avons besoin d'un nom pour une valeur donnée (fonctionnel) ou bien si c'est vraiment une variable qu'il nous faut (impératif). Pour finir, instancier un objet ne nécessite pas l'utilisation du mot-clé `new`.

Bon il y a quand même quelque chose qui ne va pas dans le code ci-dessus, qu'est-ce que c'est que cette classe `Point` avec un constructeur qui prend deux `Double` en paramètre ? La classe `java.awt.Point` prend deux `Int` et n'est donc pas compatible.

Il faut créer une nouvelle classe `Point` :

```
data class Point(val x : Double, val y : Double)
```

Et... c'est tout pour la classe Point :) Utiliser `data class` plutôt que `class` a à peu près le même effet que l'annotation `@Data` de Lombok. Dans ce cas-ci, comme `x` et `y` sont des valeurs et pas des variables, il ne sera pas possible de mettre à jour `x` et `y` dans les instances de `Point`, ce qui est exactement ce qu'on veut. Notez enfin qu'une classe en Kotlin est publique par défaut. Il nous reste à créer une implémentation de l'interface Generator qui générera des points en suivant la fonction sinus.

Les points à générer doivent répondre à plusieurs exigences : suivre la courbe du sinus, et avoir un bruit aléatoire.

```
import kotlin.math.sin
import kotlin.math.PI

class SineGenerator(
    private val amplitude : Double, private val verticalShift : Double,
    private val period : Double, private val phaseShift : Double
) : Generator {

    override fun generate(x : Double) =
        amplitude * sin(x / period * 2 * PI * phaseShift) + verticalShift

}
```

J'ai profité de cette classe pour introduire une autre fonctionnalité interessante de Kotlin : Pour définir des fonctions courtes, on peut donner directement une expression après un signe `=`. Dans ce cas, il n'est pas nécessaire de préciser le type de retour de la fonction : le compilateur va l'inférer automatiquement. Pour préciser qu'on implémente une méthode abstraite, on utilise le mot-clé `override` plutôt que l'annotation Java `@Override`.

Kotlin introduit certaines fonctions et constantes dans un package `kotlin`. On préfère en général les utiliser quand elles sont disponibles plutôt que leurs équivalents Java, mais nous verrons plus loin un contre-exemple.

Décorateurs !

Avez-vous déjà entendu parler de ce design pattern ? Il permet d'ajouter une fonctionnalité à une classe existante sans interférer avec son fonctionnement. C'est très utile, par exemple, pour ajouter des instructions de log à une classe. Mais dans notre cas nous allons nous en servir pour ajouter du bruit à un générateur existant. Dans un premier temps, je n'avais pas compris que le bruit suivait une loi normale, et j'avais donc créé un générateur ajoutant un bruit uniforme :

```
import kotlin.random.Random

class UniformNoiseGenerator(
    private val generator : Generator,
    private val maxNoise : Double
) : Generator {

    override fun generate(x : Double) : Double {
        return generator.generate(x) + Random.nextDouble() * maxNoise * 2 - maxNoise
    }

}
```

Le premier argument du constructeur prend un `Generator` dont la classe est elle-même une implémentation. C'est typique pour les décorateurs. On utilise également `Random.nextDouble` qui est fourni par la librairie standard de Kotlin. Il ne faut pas instancier d'objet `Random` , ce qui est conforme à l'objectif général de concision poursuivi par Kotlin.

Mais nous voulons un bruit qui suit une distribution normale. Pas de problème, implémentons un autre décorateur qui utilisera java.util.Random.nextGaussian... qui n'est pas dans la librairie standard de Kotlin.

Kotlin brille par sa concision et sa compatibilité avec Java.

```
import java.util.Random

class GaussianNoiseGenerator(
    private val generator : Generator,
    private val stddev : Double
) : Generator {

    private val random = Random()

    override fun generate(x : Double) =
        generator.generate(x) + random.nextGaussian(0.0, stddev)

}
```

Exporter le résultat

Avant de construire la fonction `main`, j'aimerais implémenter la fonctionnalité qui permet d'exporter le résultat au format supporté par gnuplot. L'idée étant d'afficher les points une fois ces derniers créés.

Alors aujourd'hui j'utilise gnuplot mais demain j'utiliserai peut-être R (qui aujourd'hui est capable de lire le format gnuplot mais qui sait si cette fonctionnalité ne disparaîtra pas demain ?). Enfin soit, c'est mon article, j'écris une interface dans un pur esprit d'over-engineering si ça m'amuse !

```
import java.io.File

interface Exporter {

    fun export(point : Point) : String

    fun export(points : List<Point>) : String

    fun export(points : List<Point>, file : File) =
        file.writeText(export(points))

}
```

... et l'implémentation qui va avec :

```
class GnuplotExporter : Exporter {

    override fun export(point : Point) = "${point.x}, ${point.y}"

    override fun export(points : List<Point) =
        points.joinToString("\n") { export(it) }

}
```

Remarquez que l'implémentation ne fait pas du tout référence à `java.io.File`, l'interface gérant totalement ce qui concerne les fichiers.

Lier la mayonnaise

Il ne nous reste plus qu'à écrire notre fonction `main` ! En kotlin, il suffit de déclarer une fonction qui s'appelle `main`. Elle peut prendre en argument une liste de `String`, ou bien rien du tout. Pour faire simple on va tout coder en dur et donc utiliser la deuxième option.

```
import java.io.File

fun main() {
    val sineGenerator = SineGenerator(1.0, 0.0, 1.0, 0.0)
    val generator = GaussianNoiseGenerator(sineGenerator, 0.3)
    val exporter = GnuplotExporter()

    val points = generator.generatePoints(0.0, 1.0, 100)
    exporter.export(points, File("data"))
}
```

Si on ouvre le fichier data après avoir exécuté ce programme, on trouvera 100 points dedans, au format gnuplot :-)

Afficher les points avec gnuplot

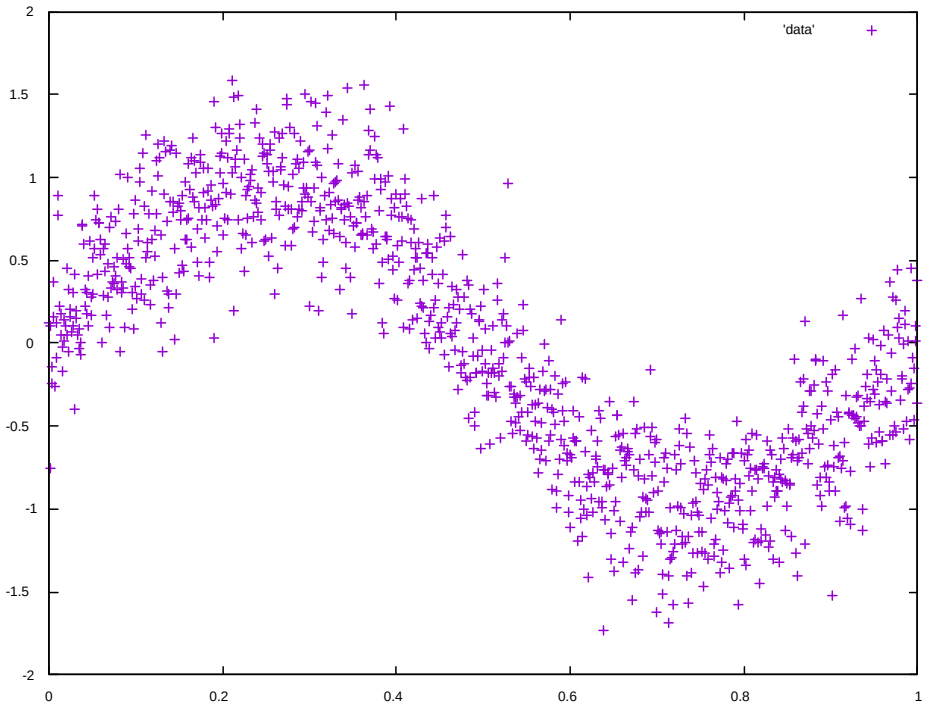
Le plan initial était d'utiliser gnuplot après avoir généré les points avec le programme Kotlin pour afficher les points. Mais mon côté paresseux a vite repris le dessus. On peut facilement adapter la fonction `main` ci-dessus pour lancer gnuplot et afficher les points directement à partir du fichier `data` qui vient d'être créé.

```
import java.io.File

fun main() {
    val sineGenerator = SineGenerator(1.0, 0.0, 1.0, 0.0)
    val generator = GaussianNoiseGenerator(sineGenerator, 0.3)
    val exporter = GnuplotExporter()

    val points = generator.generatePoints(0.0, 1.0, 100)
    exporter.export(points, File("data"))

    ProcessBuilder("gnuplot", "-persist", "-e", "plot 'data'")
        .start()
        .waitFor()
}
```



Le résultat de notre labeur

Conclusion

À travers cet article, en plus de créer un générateur de points adapté à l'apprentissage automatique, nous avons exploré les bases de Kotlin et vu le concept architectural des décorateurs.

Libre à vous d'ajouter d'autres générateurs, certaines questions n'ont d'ailleurs pas été abordées. Voici une petite piste : comment créer un générateur qui a des zones blanches dans lesquelles aucun point ne peut être généré ?

Pratiques de développement agile

La maîtrise de l'agilité
n'est pas seulement une
compétence, mais un art,
[...] fondement d'une
véritable excellence
en ingénierie logicielle.

Au travers ces deux articles d'experts, cette section aborde l'importance des méthodes de développement agile, en se concentrant sur l'efficacité du déploiement et la capacité à s'adapter rapidement aux changements.

- **Déployer son application Spring Boot dans le Cloud**. Cet article fournit un guide détaillé sur le déploiement d'applications Spring Boot dans des environnements cloud, en insistant sur les meilleures pratiques et les stratégies pour optimiser la performance et la sécurité.

- **Déployez comme un pro : le guide ultime pour Dockeriser votre application Node.js !** Cet article est un manuel détaillé pour intégrer Docker dans le développement d'applications Node.js. Il explique comment développer une API avec Fastify, préparer un Dockerfile pour créer une image Docker, et convertir cette image en un conteneur fonctionnel, tout en discutant des choix stratégiques et des meilleures pratiques dans l'usage de Docker.

> **Plus [une image Docker] est légère, plus vite l'application sera disponible et moins cela nous coûtera en termes de stockage.**

Déployer son application Spring Boot dans le Cloud

Par Guillaume FOURNY.

Le déploiement des applications Spring dans un environnement Cloud devient la norme. Voici comment le faire via diverses optimisations et en garantissant un niveau de sécurité élevé.

Beaucoup d'entreprises aujourd'hui se tournent ou se sont tournées vers des Cloud Provider public ou privé pour y déployer son parc applicatif. Ce choix est souvent poussé par l'envie de se débarrasser de ces serveurs physiques, ne plus se préoccuper de la disponibilité des applications et au passage, faire des économies !

Il y a tout de même quelques règles de l'art à respecter afin de garantir l'efficience et la sécurité des conteneurs que nous déployons sur nos pods/nodes.

Nous ne parlerons pas ici d'image native, elles répondent à des besoins très spécifiques et limités. La compilation native est très bien pour le serverless, les Cloud Functions, mais n'apportent pas le niveau de performance que nous procure la JVM.

Le mauvaise exemple

Commençons tout de suite par ce qu'il ne faut PAS faire :

```
FROM eclipse-temurin:17-jdk-alpine
VOLUME /tmp
COPY target/*.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Pourquoi ? Pour les raisons suivantes :

- Mon image de base pèse à elle seule près de 200 Mo ! Cela représente déjà une belle application Spring Boot.
- Mon image ne sera pas rootless.
- Elle embarque des librairies dont je n'ai pas besoin.
- La commande de lancement ne permet pas de bénéficier des optimisations de Spring effectuées à la compilation (layers).

Une piqûre de rappel est donc nécessaire. Une image docker n'est pas une VM ! J'ai trop souvent vu dans des Dockerfile l'installation de tool ou l'utilisation d'une image ENORME (~800Mo ça fait chère sur un Artifactory) embarquant ces différents tool (curl/wget/nano/vi etc).

Pour faire simple, une image docker, c'est notre application et uniquement celle-ci. Plus cette dernière est légère, plus vite l'application sera disponible et moins cela nous coûtera en termes de stockage (et c'est bon pour l'environnement) !

Le bon exemple

Allez, voyons comment améliorer tout ça, et là pas de miracle, on s'en réfère à la documentation officielle

```
FROM eclipse-temurin:17-jre as builder
WORKDIR application
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} application.jar
RUN java -Djarmode=layertools -jar application.jar extract

FROM eclipse-temurin:17-jre
WORKDIR application
COPY --from=builder application/dependencies/ ./
COPY --from=builder application/spring-boot-loader/ ./
COPY --from=builder application/snapshot-dependencies/ ./
COPY --from=builder application/application/ ./
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
```

L'idée générale ici est de découper notre création d'images en deux parties. La première dite "builder" va nous permettre de construire notre image final dénuée de tout ce qui est inutile !

Il va s'en dire qu'en modifiant un peu les images de bases utilisées, nous pouvons choisir notre JVM, ainsi qu'une image dite "distroless" afin que sont poids soit proche du poids de notre application. Fini le superflu, ici, vous pouvez oubliez les accès root (nous avons une image rootless), pas de curl/wget/nano ou autre il n'y a rien d'autre que votre JVM et votre application.

Nous bénéficions en plus de l'optimisation des layers Docker en utilisant les layers Spring, nous permettant de gagner du temps au rebuild de l'image et au temps de démarrage (c'est toujours ça de pris).

Il manque encore quelque chose...

En tant que développeur Java que nous sommes, nous avons une "contrainte" supplémentaire comparée à d'autres langages, la JVM !

Pour faire court, la JVM n'est pas magique, je vous épargne tous les détails, mais je vous mets des liens pour appuyer l'importance de ce que je vous présente.

Configuration de la Heap

Ce n'est pas une option ! Et on ne le fait pas au hasard non plus. Il existe un outil qui vous permet de calculer précisément vos options de JVM en fonction des ressources disponibles et du nombre de classes chargées par l'application.

Vous avez deux possibilités, soit vous clonez le repo git suivant :

- https://github.com/cloudfoundry/java-buildpack-memory-calculator

Ou alors, vous faites comme moi et vous utilisez l'image docker :)

- https://hub.docker.com/r/demaniak/java-buildpack-memory-calculator

Attention, si vous allouez plus de mémoire que nécessaire, vous risquez d'avoir régulièrement un full GC et des temps de pauses considérables (plusieurs secondes). Et voilà ma transition pour le chapitre suivant !

Configuration du Garbage Collector

Le choix du GC n'est pas une option non plus !

À tous ceux qui pensent que le G1 est le GC par défaut depuis Java 11, vous avez tort. Pour s'en rendre compte, un petit tour du côté du code source de la hotpost :

- https://github.com/openjdk/jdk/blob/master/src/hotspot/share/runtime/os.cpp#LL1710C8-L1710C8

- https://github.com/openjdk/jdk/blob/master/src/hotspot/share/gc/shared/gcConfig.cpp#L98

Que nous apprennent ces deux liens ? Grosso modo, si vous avez settez moins de 2 Go de RAM et moins de 2 CPU sur votre pod/node, votre application utilise le SerialGC dans le cas où vous n'auriez pas précisez le GC à utiliser dans vos options de JVM.

Ce n'est pas une si mauvaise chose, car la documentation d'Oracle nous préconise bien le SerialGC sur du monothread, ce qui est normal, car les autres ont été pensés multithread. Il faut tout de même l'avoir en tête pour ne pas se faire surprendre par un comportement inattendu entre l'environnement de recette et l'environnement de production par exemple :)

Qui écrit encore des Dockerfile ?

Il faut maintenir tous ces Dockerfile. Cela peut vite devenir chronophage et prendre du temps s'il faut vérifier qu'il existe une nouvelle version de l'image de base ou un patch sur la JVM. Heureusement le plugin spring-boot est là pour nous !

Voici un exemple de son usage en condition réelle :

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <imageBuilder>paketobuildpacks/builder:full</imageBuilder>
        <image>
            <name>${ARTIFACTORY_DOCKER_REGISTRY}/${ARTIFACTORY_DOCKER_DIR}/gfr-referential:${ARTIFACTORY_DOCKER_TAG}</name>
            <env>
                <BP_JVM_VERSION>${java.version}</BP_JVM_VERSION>
                <BPL_DEBUG_ENABLED>false</BPL_DEBUG_ENABLED>
                <BPE_DELIM_JAVA_TOOL_OPTIONS xml:space="preserve"> </BPE_DELIM_JAVA_TOOL_OPTIONS>
                <BPE_APPEND_JAVA_TOOL_OPTIONS> -Dfile.encoding=UTF-8</BPE_APPEND_JAVA_TOOL_OPTIONS>
            </env>
            <bindings>
                <binding>${project.basedir}/bindings:/platform/bindings/ca-certificates</binding>
            </bindings>
            <publish>true</publish>
        </image>
        <layers>
            <enabled>true</enabled>
        </layers>
        <docker>
            <publishRegistry>
                <username>${ARTIFACTORY_DIOD_USERNAME}</username>
                <password>${ARTIFACTORY_DIOD_PASSWORD}</password>
            </publishRegistry>
        </docker>
    </configuration>
</plugin>
```

Ce plugin nous permet de créer une image Docker Cloud Native sans avoir recours à un Dockerfile. Pas besoin non plus d'un Dockerfile pour copier un certificat et l'intégrer aux certificats connus par la JVM via la commande keytool, ici, c'est prévu par les bindings ;)

Par Cloud Native, on entend :

- Rootless
- Optimisation des layers
- Optimisation de la JVM

Le plugin s'appuie sur l'utilisation de buildpack et de paketo pour construire une image sécurisée, optimisée et toujours à jours (par défaut maintenue par paketo).

En utilisant ce plugin, vous n'avez pas besoin d'utiliser le calculateur des options de JVM, car il est **inclus** dans l'image final ! Vous aurez seulement besoin de spécifier le GC que votre application doit utiliser.

Optimiser l'application Spring Boot

Quelques optimisations existent pour gagner en temps de démarrage et réduire la consommation de ressources.

Ces optimisations sont préconisées dans la documentation Google Cloud Run notamment :

- spring-content-indexer
  - Évite l'analyse des classes au démarrages, consommateur en terme d'I/O disque. Les beans sont indexés à la compilation

- spring.main.lazy-initialization=true
  - Initialisation des beans uniquement lors du premier usage

- Utiliser WebFlux / programmation réactive (même avec l'arrivée de Loom)
  - https://filia-aleks.medium.com/project-loom-with-spring-boot-performance-tests-c007e0e411c8

CRaC !

Vous pouvez oublier l'indexation des beans à la compilation ainsi que l'initialisation différée des beans avec l'arrivée de CRaC (Coordinated Restore at Checkpoint).

CRaC est un projet de l'OpenJDK et va changer radicalement notre manière de déployer une application Java en mode JVM. En quelques mots, ce projet permet de démarrer une application Java en moins de 100 ms !

Là, vous pensez sûrement que ça n'est pas demain la veille que l'on aura ça. Détrompez-vous, vous l'avez peut-être déjà utilisé via les fonctions Lambda d'AWS, sous sa dénomination SnapStart !

L'autre bonne nouvelle, c'est que le framework Spring 6.1 est compatible avec CRaC, une démo est disponible ici : https://github.com/sdeleuze/spring-boot-crac-demo

Ci-dessous, une liste de JDK embarquant déjà CRaC :

- Amazon Corretto
- IBM Semeru
- Azul Zulu
- Azul Platform Prime

Plusieurs conférences ont été données sur ce sujet dont une lors du dernier Devoxx France : https://www.sfeir.dev/back/les-temps-forts-de-devoxx-2023/

Déployez comme un pro : le guide ultime pour Dockeriser votre application Node.js !

Par Alexandre MOEVI.

Découvrez Docker en toute sérénité ! Grâce à ce guide étape par étape, transformez votre application Node.js en un conteneur Docker.

Docker est un logiciel qui permet de **conteneuriser** des applications, c'est-à-dire d'emballer une application et ses dépendances dans un **conteneur**. Ce conteneur peut ensuite être exécuté de manière uniforme sur n'importe quelle machine, garantissant que l'application fonctionnera toujours de la même manière, indépendamment de l'endroit où elle est déployée.

Prérequis

Avant de plonger dans le vif du sujet, assurez-vous que Node.js et npm sont bien installés et prêts à être utilisés pour le développement de notre application. De même, vérifiez que Docker est également installé pour la conteneurisation de notre application.

Vous pouvez vérifier que tout est correctement installé en utilisant les commandes suivantes.

```
$ npm --version
9.5.0
$ node --version
v18.14.2
$ docker --version
Docker version 19.03.12, build 48a66213fe
```

Développement d'une API avec fastify

Avant de démarrer le développement avec Fastify, mettons en place l'environnement de notre projet.

```
$ mkdir fastify-api-project
$ cd fastify-api-project
$ npm init -y
$ npm install fastify
```

Avec `mkdir fastify-api-project`, nous créons un répertoire spécifique pour notre application. `cd fastify-api-project` nous permet d'entrer dans ce répertoire, c'est l'endroit où tout notre développement se déroulera. `npm init -y` démarre l'initialisation d'un projet Node.js, générant un fichier essentiel: `package.json`. Pour finir, `npm install fastify` installe Fastify, le framework que nous utiliserons pour développer notre API.

Étape suivante, créons le fichier `index.js` dans lequel nous allons ajouter ce code.

```
const Fastify = require('fastify');
const fastify = Fastify({ logger: true });

const PORT = parseInt(process.env.PORT, 10) || 3000;
const HOST = process.env.HOST || 'localhost';

fastify.get('/', async (request, reply) => {
    reply
        .code(200)
        .header('Content-Type', 'application/json; charset=utf-8')
        .send({ site: 'sfeir.dev', article: 'build node js app with docker' });
});

fastify.listen({ host: HOST, port: PORT }, (err, address) => {
    if (err) {
        console.error(err)
        process.exit(1)
    }
});
```

Ce code met en place un service renvoyant des données au format JSON. Notons la présence des variables d'environnement `HOST` et `PORT`, qui seront utiles au moment de la conteneurisation.

Dès que vous avez enregistré le fichier `index.js`, l'API est prête à être testée. En exécutant `node index.js` dans votre terminal, vous devriez pouvoir accéder à `http://localhost:3000` dans votre navigateur et voir la réponse JSON.

Avec notre application en place, voyons comment la mettre dans un conteneur Docker.

Création de l'image Docker

Pour ce faire, nous allons créer un fichier nommé `Dockerfile`. Ce fichier contiendra les instructions permettant à Docker de construire une image de notre application.

```
FROM node:18

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY --chown=node:node . .

EXPOSE 3000

ENV HOST=0.0.0.0 PORT=3000

CMD ["node", "index.js"]
```

Passons en revue les instructions, ligne par ligne.

• `FROM node:18` indique à Docker de se baser sur l'image officielle de Node.js version 18. C'est la fondation sur laquelle notre conteneur sera construit. Cette version de Node.js fournira l'environnement d'exécution pour notre application.

• `WORKDIR /app` définit le répertoire de travail dans le conteneur à /app. Toutes les commandes qui suivent s'exécuteront dans ce répertoire.

• `COPY package*.json ./` copie les fichiers `package.json` et `package-lock.json` (s'il existe) du répertoire source (notre machine) vers le répertoire de travail du conteneur. Elle est essentielle pour installer les dépendances nécessaires à notre application.

• `RUN npm install` exécute la commande npm install dans le conteneur.

• `COPY --chown=node:node . .` copie le reste des fichiers et dossiers de notre application dans le conteneur. L'option `--chown=node:node` garantit que les fichiers sont possédés par l'utilisateur `node`, augmentant ainsi la sécurité en évitant de fonctionner avec des permissions superutilisateur.

• `EXPOSE 3000` indique à Docker que notre futur conteneur écoutera sur le port 3000. C'est le port sur lequel l'API Fastify fonctionnera.

• `ENV HOST=0.0.0.0 PORT=3000` définit les variables d'environnement `HOST` et `PORT` à utiliser dans notre service. L'utilisation de `0.0.0.0` pour `HOST` permet à l'application d'être accessible depuis l'extérieur du conteneur.

• `CMD ["node", "index.js"]` spécifie la commande qui sera exécutée lorsque le conteneur sera lancé. Ici, elle démarrera notre application avec Node.js.

La commande suivante construit une image Docker en se basant sur notre Dockerfile et lui donne le nom sfeir-dev .

```
docker build -t sfeir-dev .
```

La commande docker run est la clé pour transformer une image Docker en conteneur éxécutable, assurant la portabilité de votre application Node.js.

Pour s'assurer que l'image a été correctement construite et est maintenant disponible, on peut utiliser `docker images`.

```
$ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
sfeir-dev                     latest    d1b90f727fbc   6 seconds ago   1.1GB
```

Ainsi, notre image est prête à être déployée et exécutée sous forme de conteneur.

De l'image au conteneur

Nous y sommes presque. Nous allons utiliser l'image que nous venons de construire et la transformer en conteneur exécutable. Pour ce faire, nous utiliserons la commande docker run.

"docker ps" : la commande clé pour afficher la liste des conteneurs en cours d'exécution (et leurs identifiants).

```
$ docker run -d -p 3000:3000 sfeir-dev
d3fae3a516aa6b652574b6fc726156479a1e6196234838b81cb5e467529124c0
```

Arrêtons nous sur cette commande :

• `docker run` est la commande de base pour démarrer un conteneur à partir d'une image Docker.

• `-d` est l'argument pour activer le mode détaché. Cela signifie que Docker exécutera le conteneur en arrière-plan. Sans cette option, le conteneur serait exécuté au premier plan et bloquerait le terminal.

• `-p 3000:3000` indique à Docker d'effectuer une liaison de port. Le format est `<port sur l'hôte>:<port sur le conteneur>`. Cela signifie que le port 3000 du conteneur sera accessible via le port 3000 de la machine locale. Dans le contexte de notre API, cela signifie que si notre application Fastify écoute sur le port 3000 à l'intérieur du conteneur, nous pouvons y accéder en utilisant le port 3000 sur notre machine.

• `sfeir-dev` est le nom de l'image que nous avons créée précédemment. Docker démarrera un conteneur basé sur cette image.

• Une fois que cette commande est exécutée, votre API Fastify tourne maintenant à l'intérieur d'un conteneur Docker. Vous pouvez le vérifier en accédant à http://localhost:3000 dans votre navigateur ou en utilisant un outil comme `curl` depuis votre terminal. Dans le même temps, Docker fournit également en sortie de la commande un identifiant unique pour ce conteneur.

• Si vous n'avez pas gardé cet identifiant sous la main, pas de panique ! La commande `docker ps` est là pour nous aider. Elle affiche la liste des conteneurs en cours d'exécution, y compris leurs identifiants.

```
$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED
d3fae3a516aa   sfeir-dev   "docker-entrypoint.s…"   9 seconds ago
```

```
 STATUS         PORTS                    NAMES
 Up 8 seconds   0.0.0.0:3000->3000/tcp   vibrant_nobel
```

De nombreuses commandes Docker utilisent l'identifiant comme argument afin d'interagir avec le conteneur spécifique.

Par exemple, la commande `docker logs` permet de consulter les logs de l'application.

```
$ docker logs d3fae3a516aa
{"level":30,"time":1696762134174,"pid":1,"hostname":"d3fae3a516aa","msg":"Server listening
{"level":30,"time":1696762807374,"pid":1,"hostname":"d3fae3a516aa","reqId":"req-1","req":{
{"level":30,"time":1696762807381,"pid":1,"hostname":"d3fae3a516aa","reqId":"req-1","res":{
```

```
at http://0.0.0.0:3000"}
"method":"GET","url":"/","hostname":"localhost:3000","remoteAddress":"172.17.0.1","remotePort":
"statusCode":200},"responseTime":5.7870680000633,"msg":"request completed"}
```

```
58620},"msg":"incoming request"}
```

Suppression des ressources Docker

Pour arrêter ou supprimer le conteneur, vous aurez besoin de son identifiant. Arrêter et supprimer sont deux opérations distinctes.

Arrêter un conteneur met fin à son exécution, mais il reste présent sur le système jusqu'à ce qu'il soit explicitement supprimé. Pour arrêter le conteneur, utilisez la commande `docker stop`. Si vous souhaitez le supprimer après l'avoir arrêté, utilisez `docker rm`.

```
$ docker stop d3fae3a516aa
d3fae3a516aa
$ docker rm d3fae3a516aa
d3fae3a516aa
```

Si vous décidez de supprimer l'image Docker elle-même, vous pouvez le faire en utilisant l'identifiant de l'image ou son nom. Avant cela, assurez-vous que tous les conteneurs associés à cette image soient arrêtés et supprimés. Pour supprimer l'image, utilisez la commande `docker rmi` .

```
$ docker rmi sfeir-dev
Untagged: sfeir-dev:latest
Deleted: sha256:d1b90f727fbcf466c986836ada75a3350921e11d6d82dd77b1449a8a9bc9d9d8
```

Larguez les amarres !

Nous avons construit notre premier conteneur Docker. aCet article s'est concentré sur l'essentiel mais Docker offre un océan de fonctionnalités et de détails qui méritent une exploration approfondie.

Pourquoi avons-nous utilisé `0.0.0.0` comme `HOST` ? Qu'arriverait-il si nous avions utilisé `localhost` à la place ?

Si vous avez déjà travaillé avec Docker, vous avez peut-être rencontré les images "alpine", reconnues pour leur légèreté. Alors, pourquoi ne pas avoir choisi `node:18-alpine` ?

En parlant de taille, l'instruction `COPY` transfère tout le contenu du répertoire dans le conteneur. Cela peut inclure des fichiers superflus ou sensibles.

Toutes ces interrogations ne sont qu'un avant-goût des nombreuses astuces et meilleures pratiques que nous pourrions explorer sur sfeir.dev, plongeant un peu plus dans l'univers de Docker.

Sécurité et fiabilité

Chez SFEIR, la sécurité et la qualité ne sont pas optionnelles mais essentielles.

La sécurité et la fiabilité sont des piliers essentiels dans le développement d'applications Back-End robustes et sécurisées. Voici deux articles de nos experts que nous souhaitions vous partager.

- **Développement web sécurisé - découvrez les incontournables avec l'OWASP Top Ten.** Cet article souligne l'importance de la connaissance des risques de sécurité les plus courants, tels qu'identifiés par l'OWASP Top Ten, et comment les atténuer efficacement.

- **Gérer facilement ses clés SSL avec Spring Boot 3.1.** Cet essai décrit l'introduction des SSL Bundles dans Spring Boot 3.1, facilitant la configuration et la gestion des clés SSL pour divers composants.

Cette fonctionnalité simplifie l'implémentation de connexions sécurisées HTTPS, TLS et mTLS dans des applications Spring Boot.

Avec l'expertise OWASP et Spring Boot 3.1, SFEIR redéfinit la sécurité des applications Back-End.

Développement web sécurisé – découvrez les incontournables avec l'OWASP Top Ten

Par Mathieu VANDENNEUCKER.

La sécurité web évolue très rapidement, mais certaines vulnérabilités incontournables existent depuis des années et continuent d'être exploitées. Tout développeur web se doit de les connaître, et pour cela, l'OWASP Top Ten est un excellent outil !

Vous souhaitez améliorer la sécurité de vos applications web, mais ne savez pas par où commencer ? Vous avez entendu parler de l'OWASP Top Ten et souhaitez en savoir plus ? Cet article est fait pour vous ! La sécurité est un domaine incontournable dans le monde du développement informatique, et particulièrement au niveau du développement web / cloud.

Cependant, la sécurité web est un vaste sujet qui évolue à une vitesse fulgurante. De nombreuses nouvelles failles de sécurité sont découvertes chaque jour, ce qui peut sembler difficile à suivre.

Néanmoins, certaines catégories de vulnérabilités existant depuis plus de 10 ans (et parfois même plus de 20 ans) restent très répandues aujourd'hui : les injections SQL, les failles XSS, CSRF… On pourrait les appeler les incontournables. Et c'est par là qu'il faut commencer !

Par où ne pas commencer

Vous venez de rejoindre un projet et souhaitez évaluer l'état de la sécurité ? Ou l'on vous demande de renforcer la sécurité du projet sur lequel vous travaillez ? Des premières questions se posent alors : comment faire, et par où commencer ? Il n'y a pas qu'une seule réponse valide à ces questions : comme souvent, cela dépend du contexte. Mais une chose est sûre, si vous avez le sentiment que peu de choses ont été mises en place jusqu'ici au niveau de la sécurité du projet, il y a certaines démarches par lesquelles il ne vaut mieux pas commencer.

Par exemple, faire appel directement à une entreprise de ***pentests***. Un pentest, ou test de pénétration, consiste à attaquer un réseau/logiciel/application pour tenter d'y découvrir des failles et de réaliser des actions normalement non autorisées, et ce dans un certain cadre légal et un contexte donné. De nombreuses entreprises spécialisées proposent leurs services de pentests, ce qui

permet de trouver des failles qu'il n'est pas toujours évident d'identifier soi-même.

Et c'est là où réside le problème : si les pentesteurs passent du temps à identifier des failles que vous auriez pu découvrir vous-même, c'est à la fois du temps perdu pour cette équipe et de l'argent perdu pour vous. En effet, un pentest dure un nombre de jours (généralement déterminé à l'avance) et le temps que les pentesteurs passent à confirmer les vulnérabilités "évidentes", c'est du temps qu'ils ne passent pas à chercher des vulnérabilités plus avancées. Et de votre côté, cela aurait pu vous coûter moins cher d'identifier ces failles "évidentes" vous-même, plutôt que de payer un spécialiste pour le faire. Tout le monde y perd, donc !

Une autre approche non recommandée consisterait à s'inscrire directement à un programme de ***bug bounty***. Un tel programme définit un cadre dans lequel n'importe qui (ou parfois un groupe déterminé de personnes) peut, dans une certaine limite, tester la sécurité d'une application. Lorsqu'une personne découvre une faille, elle la communique à l'entreprise propriétaire du projet, et celle-ci peut alors corriger la faille et rémunérer la personne selon des critères prédéterminés, en fonction de la criticité de la faille. Le montant payé dépend donc du risque lié à la faille, plutôt que du temps passé à rechercher la faille (à l'inverse d'un pentest, comme vu plus haut). Ici, c'est surtout vous qui allez y perdre : les pentesteurs vont vous signaler des failles "évidentes", et vous allez devoir les rémunérer pour cela. Cela vous coûtera, à nouveau, plus cher que si vous les aviez identifiées et corrigées avant de rejoindre le programme de bug bounty.

Mais alors, par où commencer ? Vous l'aurez deviné au titre de l'article, l'OWASP Top Ten est une très bonne première étape, comme nous allons le voir juste après. Les pentests et programmes de bug bounty sont des outils intéressants, mais à utiliser au bon moment : une fois que le projet a atteint un certain niveau de maturité en matière de sécurité.

Qu'est-ce que l'OWASP ?

L'OWASP (Open Web Application Security Project), c'est une communauté mondiale à but non lucratif dédiée à la sécurité des applications web.

Son objectif est d'**améliorer la sécurité des logiciels,** par de la sensibilisation et via différents projets.

Ces projets sont très variés : on y retrouve Dependency-Track - un outil de suivi des vulnérabilités dans les dépendances, Juice Shop - une application vulnérable que l'on peut utiliser pour de l'apprentissage ou de l'entraînement, Zed Attack Proxy (ZAP) - un outil d'analyse dynamique de la sécurité d'une application, et plein d'autres.

Et bien entendu, parmi ces projets, il y a l'**OWASP Top Ten** : il s'agit d'un consensus sur les risques de sécurité les plus critiques des applications web. Concrètement, l'OWASP Top Ten est divisé en dix catégories de risques, triées par fréquence d'apparition. Ces catégories proviennent d'analyse de données de vulnérabilités de nombreuses entreprises, mais également d'un sondage de la communauté.

Chaque catégorie est en fait un ensemble de **CWE** - Common Weakness Enumeration. Une CWE est un type de faiblesse logicielle, c'est-à-dire une caractéristique d'une application pouvant contribuer à introduire une vulnérabilité. Par exemple, le fait d'utiliser des identifiants hardcodés, de ne pas hacher les mots de passes, de ne pas valider les URLs provenant de l'extérieur, etc.

Pour chaque type de faiblesse (CWE), on peut retrouver des vulnérabilités concrètes sur des applications existantes : il s'agit de **CVE** - Common Vulnerabilities and Exposures. Une CVE est une vulnérabilité découverte dans une application. Par exemple, la CVE-2021-44228 (Log4shell) est une vulnérabilité bien connue découverte sur la librairie Log4j en décembre 2021, et liée notamment à la CWE-502 (désérialisation de données non fiables).

En résumé, chacune des 10 catégories de l'OWASP Top Ten est composée d'un ensemble de types de faiblesses (CWE), chaque type de faiblesse étant lié à différentes vulnérabilités (CVE) découvertes sur des applications réelles.

L'OWASP Top Ten : les incontournables

Parcourons maintenant ces différentes catégories, et certaines des CWE qui les composent. Pour chacune des CWE, vous retrouverez plus d'informations (en anglais) sur le lien qui y est lié.

A01:2021-Broken Access Control

Photo by Kirk Thornton / Unsplash

Il s'agit de tout ce qui concerne le contrôle d'accès: pages ou APIs non protégées (ou pas assez), élévation de privilèges, contournement des contrôles d'accès...

• **Path traversal :** vulnérabilité exploitable lorsqu'un input utilisateur est utilisé dans un chemin, par exemple pour récupérer un fichier, et qu'il manque de la validation. Par exemple, pour une page */photo?path=albumId/photo44.jpg,* si les photos sont stockées dans */data/* et que l'utilisateur donne la valeur *"../etc/passwd"* (un fichier sensible sur un serveur Linux) pour le paramètre path, sans validation supplémentaire, l'utilisateur risque de pouvoir récupérer le contenu de ce fichier sensible.

• **Cross-Site Request Forgery (CSRF) :** attaque consistant à réaliser une action non souhaitée au nom d'un utilisateur, via son navigateur. Cela peut se produire si l'utilisateur clique sur un lien envoyé par l'attaquant. Pour s'en protéger, une mesure de protection anti-CSRF doit être mise en place, mais attention que cette mesure ne protège pas les requêtes GET: il faut donc s'assurer que les requêtes réalisant des actions (création, modification, suppression...) n'utilisent pas GET.

• **Open redirect :** cette attaque permet de rediriger un utilisateur vers un site malveillant, en lui faisant accéder à un lien valide d'une application légitime. Cela se produit généralement lorsqu'un site propose une page telle que */login?redirectAfterLogin=https://example.com*, et que le paramètre de l'URL n'est pas validé. C'est risqué, car si le site malveillant est une copie conforme du site original, l'utilisateur risque de ne pas se rendre compte qu'il n'est plus sur le site original, et pourrait être amené à entrer ses identifiants.

A02:2021-Cryptographic Failures

Photo by iMattSmart / Unsplash

Cela inclut les problèmes liés à la cryptographie, et au manque de cryptographie : chiffrement non forcé, utilisation d'algorithmes dépréciés, validation incorrecte de certificats...

• **Hash des mots de passes :** il est plus que recommandé de ne pas stocker les mots de passes des utilisateurs en clair dans la base de donnée, mais bien de les hacher. Cependant, même lorsque le mot de passe est haché, des risques persistent, et une protection suffisante doit être mise en place (notamment, s'assurer que l'on utilise un algorithme de hachage suffisamment bon et avec les bons paramètres).

• **Transmission d'informations sensibles en clair :** aujourd'hui, il apparaît comme une évidence qu'il faut utiliser des protocoles chiffrés (HTTPS, SFTP...) plutôt que leurs équivalents non chiffrés, étant donné que le trafic sur internet peut être écouté par de nombreuses personnes, potentiellement malveillantes. Cependant, un site utilisant HTTPS n'est pas forcément à l'abri de tout, et d'autres mesures doivent être mises en place (protection des cookies, HSTS...).

A03:2021-Injection

Photo by Sara Bakhshi / Unsplash

Tout ce qui est lié à l'utilisation de données non validées/sanitisées dans des requêtes : SQL, NoSQL, commande OS, LDAP...

• **Injection SQL :** faut-il encore la présenter ? Cette injection bien connue est permise lorsque des valeurs provenant d'utilisateurs sont insérées dans une requête SQL, sans correctement valider ces valeurs. Cela peut permettre, selon le cas, de récupérer des informations sensibles depuis la base de données, ou d'ajouter, modifier ou supprimer du contenu.

• **Cross-Site Scripting (XSS) :** cette faille se produit lorsqu'un attaquant arrive à injecter du JavaScript sur une page d'un site légitime, visitée par d'autres utilisateurs.

A04:2021-Insecure Design

Photo by Scott Blake / Unsplash

Il s'agit ici de tous les problèmes liés à l'architecture et à la conception de l'application, et non pas à son implémentation. Une implémentation peut être aussi parfaite que possible, elle ne pourra pas toujours résoudre tous les problèmes, notamment ceux liés à la conception de l'application.

• **Mauvaise compartimentalisation :** la conception d'une application doit prendre en compte le fait que des fonctionnalités ou ressources nécessitant des accès ou permissions différentes, doivent être suffisament isolées. Par exemple, il est généralement recommandé de séparer les fonctionnalités d'administration des autres fonctionnalités.

• **Se baser sur la sécurité par l'obscurité :** La sécurité par l'obscurité, c'est le fait de ne pas divulger certaines informations ou détails liés au fonctionnement des outils, librairies utilisées, dans le but d'assurer la sécurité du système. Ces informations peuvent être par exemple le fonctionnement d'un algorithme de chiffrement, ou la version du serveur web ou d'application (Tomcat, Apache httpd...). Le fait de ne pas divulguer cela n'est pas forcément un mal en soit, c'est plutôt le fait que divulguer l'information ait pour impact de diminuer la sécurité du système (cela ne doit pas être le cas).

A05:2021-Security Misconfiguration

Photo by ABEL MARQUEZ / Unsplash

On parle ici des problèmes de configuration de sécurité de logiciels, librairies et outils utilisés.

• **Headers de sécurité :** pour les requêtes web, de nombreux headers de sécurité existent, et doivent être configurés correctement pour empêcher ou limiter certains types d'attaques: X-XSS-Protection, Content-Security-Policy (CSP), Access-Control-Allow-Origin (CORS)...

• **Page d'erreur :** configurer des pages d'erreur spécifique (pour les erreurs 4xx, 5xx, etc.), c'est éviter que des informations sensibles soient divulguées à des attaquants, en cas d'erreur.

A06:2021-Vulnerable and Outdated Components

Photo by Ale Maciel / Unsplash

Les vulnérabilités, dans les dépendances. Il peut s'agir de vulnérabilités dans les librairies utilisées (incluant donc les dépendances transitives), sur l'OS, la base de donnée, le serveur d'applications, des APIs d'autres applications... S'en protéger signifie notamment faire un suivi des dépendances que l'on utilise, et des vulnérabilités découvertes dans ces dépendances. Cela peut se faire par exemple via l'outil OWASP Dependency-Track.

Mais il faut aussi évidemment éviter l'utilisation de dépendances dépréciées, ou peu/plus maintenues (plus d'informations).

A07:2021-Identification and Authentication Failures

Photo by Kelly Sikkema / Unsplash

Il s'agit de failles liées à la confirmation de l'identité des utilisateurs (l'authentification), ainsi qu'à la gestion des sessions (mécanisme permettant de savoir quel est l'utilisateur actuellement connecté).

• **Contournement de l'authentification :** avoir un mécanisme d'authentification en place, c'est bien, mais s'assurer qu'il ne soit pas possible de passer outre ce mécanisme, c'est mieux ! Il est pour cela conseillé d'utiliser des mécanismes d'authentification et de session reconnus et sécurisés.

• **Authentification multi facteurs :** le fait de ne pas utiliser plusieurs facteurs pour authentifier l'utilisateur est aujourd'hui un risque pour la sécurité de vos applications, qu'il s'agisse des comptes de vos utilisateurs, ou des comptes administrateurs. C'est un risque notamment en cas de phishing, ou de compromission du mot de passe d'utilisateurs (par exemple dans le cas où un utilisateur utilise le même mot de passe sur plusieurs sites, et que l'un d'entre eux s'est fait attaquer).

A08:2021-Software and Data Integrity Failures

Photo by Randy Fath / Unsplash

Ces vulnérabilités se produisent lorsque l'intégrité des données reçues n'est pas vérifiée. Par exemple, lorsqu'un logiciel peut être mis-à-jour, il faut s'assurer que la mise-à-jour reçue n'est pas une fausse. Ou encore, que la librairie que l'on utilise est bien l'originale et non une copie falsifiée.

• **Désérialisation de données non fiables :** sérialiser un objet (Java, C#...), c'est le transformer dans un format de donnée (json, xml, binaire...), pour pouvoir l'envoyer ou le stocker. Et désérialiser, c'est le processus inverse : retrouver les données à partir des données sérialisées. Cependant, les fonctionnalités de (dé)sérialisation sont souvent poussées, et deviennent dangereuses lorsque les données ne sont pas de confiance: attaque par déni de service (DoS), exécution de code à distance sur le serveur (RCE)...

• **Mass assignment :** de nombreux frameworks (Spring, NodeJS...) permettent de *mapper* automatiquement des paramètres GET/POST à des objets du langage. Il est cependant parfois possible d'utiliser ce mécanisme pour modifier

des valeurs qui n'étaient pas sensées pouvoir être modifiées. On peut par exemple imaginer l'exemple d'un formulaire d'inscription, recevant différents paramètres POST (username, email, password) qui sont mappés automatiquement à un objet *User côté serveur*.

Si cet objet *User* possède une propriété booléenne *isAdmin* (à false par défaut), et que l'attaquant rajoute un paramètre POST *isAdmin=true* à la requête, l'utilisateur créé risque d'être administrateur.

A09:2021-Security Logging and Monitoring Failures

Photo by Jürgen Jester / Unsplash

Cette catégorie comprend les problèmes liés au manque de logs et de surveillance de l'application et ses événements. Il est en effet important d'avoir **assez de logs utiles**, pour pouvoir par exemple identifier lorsqu'une attaque brute-force est en cours, ou lorsqu'un utilisateur tente de réaliser des actions interdites. Mais il faut aussi avoir des mécanismes en place permettant de détecter ces attaques ou actions, et d'alerter automatiquement.

La connaissance de l'OWASP Top Ten est essentielle pour contrer les vulnérabilités web en constante évolution.

A10:2021-Server-Side Request Forgery

Photo by Sivani Bandaru / Unsplash

Cette dernière catégorie ne contient qu'une seule CWE, du même nom. Une telle vulnérabilité est exploitable lorsqu'une application récupère une ressource distante via une URL fournie par l'utilisateur, et sans valider correctement cette URL (plus d'informations).

Formez-vous sur l'OWASP Top Ten

Chacune des catégorie de l'OWASP Top Ten mérite que l'on s'y intéresse, et cela nécessite donc de connaître les principales CWE de chacune d'entre elles. Nous en avons parcouru certaines, mais il y en a d'autres.

Pour se former, une bonne ressource est simplement le site officiel du projet OWASP Top Ten : https://owasp.org/www-project-top-ten/.

Vous y retrouverez des informations sur les catégories, et des liens pour s'informer sur chacune des CWE, qu'il est toujours utile de consulter : par exemple, même si vous connaissez la faille XSS, vous apprendrez peut-être d'autres techniques d'exploitation de cette faille, que vous n'aviez pas imaginé.

D'autres ressources intéressantes :

- https://cheatsheetseries.owasp.org/ : des antisèches sur de nombreux domaines de la sécurité web ;

- https://portswigger.net/web-security : des tutoriels sur diverses vulnérabilités web ;

- https://owasp.org/www-project-vulnerable-web-applications-directory/ : une liste d'applications vulnérables, que l'on peut utiliser pour s'entraîner à exploiter des vulnérabilités (une fois sur la page, ouvrez l'onglet "Online").

- https://tryhackme.com/hacktivities : des tutoriels et ateliers (payants) sur la sécurité (web, mais pas que). Notamment cet atelier sur l'OWASP Top Ten : https://tryhackme.com/room/owasptop10.

SFEIR organise également différentes formations via le SFEIR Institute et les SFEIR Schools, et une formation sur le sujet pourrait être organisée sur demande. Une SFEIR School sur l'OWASP Top Ten existe, et les supports peuvent être trouvés à cette adresse : https://github.com/sfeir-open-source/sfeir-school-web-security.

Et ensuite ?

Vous former sur l'OWASP Top Ten vous permettra de découvrir et corriger de nombreuses vulnérabilités web sur vos applications, mais aussi d'éviter d'en introduire de nouvelles.

Votre prochaine étape pourrait être de mettre en place un *secure SDLC* (Software Development Life Cycle) : intégrer la sécurité dans les différentes étapes du cycle de vie du logiciel. Le cycle de vie d'une application est divisé en différentes phases, typiquement : récupération des exigences, conception, développement, test, déploiement, maintenance. Que l'on travaille en agile ou en *waterfall,* ces phases sont bien présentes, et la sécurité doit être incluse dans chacune d'entre elles et non uniquement durant le développement ou les tests.

Pour mettre cela en place, différentes méthodologies existent : le NIST SSDF (open source), l'OWASP SAMM (open source), ou encore BSIMM (propriétaire). La première est une méthodologie dite prescriptive: elle décrit les pratiques à mettre en place, et les actions à réaliser pour y arriver. Les deux autres sont descriptives: elles décrivent les pratiques, mais pas comment y arriver concrètement. Ces dernières laissent donc plus de liberté d'implémentation, mais nécessitent également plus d'expérience du domaine.

Dans tous les cas, la mise en place d'une méthodologie de sécurité n'est pas une tâche aisée, et s'entourer de professionnels du domaine de la sécurité des applications (*AppSec*) est conseillé.

En conclusion...

Ne mettez pas la charrue avant les bœufs. De nombreux outils et techniques existent pour la sécurité, mais il faut les utiliser à bon escient, et surtout au bon moment.

L'OWASP Top Ten, c'est une très bonne première approche à la sécurité d'une application web. Ensuite, des méthodologies de développement (SDLC) peuvent être mise en place. Et une fois le projet plus mature en terme de sécurité, des actions telles que le *pentest* ou le *bug bounty* peuvent être envisagées, pour confronter la sécurité de votre application à la réalité du terrain.

Face à la très forte hausse des cyberattaques, il est indispensable de protéger ses applications. Maintenant, à vous de jouer : les défis de sécurité n'attendent que vous !

Prévenez les failles en intégrant l'OWASP Top Ten dès le début, assurant ainsi une sécurité web solide pour vos applications.

Gérer facilement ses clés SSL avec Spring Boot 3.1

Par Yves DAUTREMAY.

Spring Boot 3.1 introduit des bundles SSL pour gérer facilement les clés SSL entre les composants. Les bundles permettent notamment de configurer un serveur en HTTPS ou d'utiliser RestTemplate avec TLS ou mTLS.

Spring Boot 3.1 a apporté quelques nouveautés intéressantes. Parmi elles, une gestion des clés SSL cohérente entre les différents composants.

SSL Bundle

La notion de SSL bundle permet de configurer des éléments SSL et de pouvoir ensuite les appliquer à d'autres composants devant communiquer à travers une connexion sécurisée.

Configurer un bundle SSL est très simple et se fait dans le fichier *application.yml*. Deux cas de figure sont supportés

Avec un Keystore

Si les clés SSL sont déjà dans un keystore

```yaml
spring:
  ssl:
    bundle:
      jks:
        mybundle:
          key:
            alias: "application"
          keystore:
            location: "classpath:application.p12"
            password: "secret"
            type: "PKCS12"
          truststore:
            location: "classpath:server.p12"
            password: "secret"
```

On configure ici un bundle JKS (Java KeyStore) qu'on appelle mybundle. C'est par ce nom qu'on le référencera plus tard.

On retrouve aussi les propriétés standards du keystore. On peut aussi configurer un trustore si on veut faire du TLS.

Avec une paire de clés SSL

Si l'on possède directement une paire de clés publique/privée au format PEM

```yaml
spring:
  ssl:
    bundle:
      pem:
        mybundle:
          keystore:
            certificate: "classpath:application.crt"
            private-key: "classpath:application.key"
          truststore:
            certificate: "classpath:server.crt"
```

On configure ici un bundle PEM appelé mybundle. On référence directement les fichiers de clé.

Utilisation des bundles

Spring boot configure automatiquement un bean SslBundles qui va nous permettre de travailler avec nos bundles, et notamment récupérer le SslContext.

```
@Component
public class MyComponent {

    public MyComponent(SslBundles sslBundles) {
        SslBundle sslBundle = sslBundles.getBundle("mybundle");
        SSLContext sslContext = sslBundle.createSslContext();
        // do something with the created sslContext
    }

}
```

Intégration avec des composants standards

Une fois un bundle configuré, il devient très facile de l'appliquer sur des technologies prises en charge par Spring boot.

Configurer le serveur en HTTPS

Une seule ligne de configuration permet de passer son serveur en HTTPS

```yaml
server:
  port: 8443
  ssl:
    bundle: "mybundle"
```

Utiliser RestTemplate en TLS (ou mTLS)

Il suffit d'appliquer le bundle SSL lors de la création du RestTemplate.

Si le bundle contient un truststore avec le certificat du serveur on pourra faire du TLS. S'il contient un keystore dont le certificat est connu du serveur, on pourra faire du mTLS.

```java
@Service
public class MyService {

    private final RestTemplate restTemplate;

    public MyService(RestTemplateBuilder restTemplateBuilder, SslBundles sslBundles) {
        this.restTemplate = restTemplateBuilder.setSslBundle(sslBundles.getBundle("mybundle")).build();
    }
}
```

Se connecter à Redis en TLS

De la même manière, si on a un bundle avec un truststore, une simple configuration permet de se connecter à Redis en TLS

```yaml
spring:
  data:
    redis:
      ssl:
        enabled: true
        bundle: "mybundle"
```

Se connecter à MongoDB en TLS

On applique encore une fois tout simplement le bundle

```yaml
spring:
  data:
    mongodb:
      uri: "mongodb://user:secret@mongoserver1.example.com:27017,mongoserver2.example.com:23456/test"
      ssl:
        enabled: true
        bundle: "mybundle"
```

Conclusion

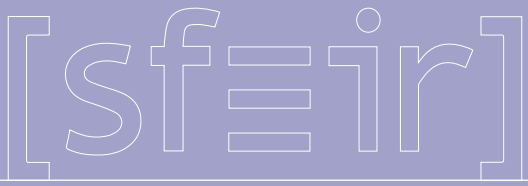
Les bundles SSL permettent d'intégrer facilement les problématiques de connexion sécurisées avec les composants Spring Boot existants, ainsi que pour des utilisations custom.

sources

Documentation Spring Boot
https://docs.spring.io/spring-boot/docs/3.1.0/reference/html/features.html#features.ssl

Optimisation et automatisation

L'optimisation et
l'automatisation propulsent
la performance et l'efficacité
du développement
Back-End avec simplicité.



Au travers ces deux articles d'experts, cette section met en évidence divers outils et techniques qui peuvent être utilisés pour optimiser et automatiser les processus de développement Back-End.

- **Simplifier la transformation d'objets avec AutoMapper.** L'article présente AutoMapper, un outil puissant pour simplifier la transformation d'objets, et comment il peut être utilisé pour augmenter l'efficacité du développement.

- **Au-delà du "with" : les gestionnaires de contexte.** Cet article explique les gestionnaires de contexte en Python, mettant en lumière l'efficacité de l'instruction with pour gérer les ressources et les exemples pratiques d'utilisation.

Simplification de la transformation d'objets avec AutoMaper et les gestionnaires de contexte en Python.

Simplifier la transformation d'objets avec AutoMapper

Par Gaëlle CHALOT.

bibliothèque .NET

Présentation et cas pratique d'utilisation de la bibliothèque .NET de référence pour le mapping d'objet.

La conception du modèle de données d'une application requiert souvent une séparation des objets en deux types distincts : les **DAO** (Data Access Object), qui incarnent les données brutes, et les **DTO** (Data Transfer Object), également appelés ViewModel dans le cadre du modèle **MVC** (Modèle-Vue-Contrôleur). Ces derniers définissent la structure des données à échanger entre les différentes couches de l'application. La couche de service, chargée de la logique métier, est responsable de la transformation de ces objets DAO en DTO. Grâce à la bibliothèque **AutoMapper**, cette tâche potentiellement laborieuse peut être entièrement automatisée.

Mise en place du projet

Supposons que l'objectif soit de créer une API pour la gestion de blogs. Trois entités principales seraient nécessaires : **les blogs, les auteurs et les articles** (ou "posts"). Le projet serait conçu en utilisant le framework **ASP.NET Core et l'ORM** (Object-Relational Mapping) **Entity Framework Core** pour l'accès à la base de données.

Réalisation du modèle de données

Nous commencerons par définir nos DAO.

```
public abstract class Entity
{
    public Guid Id { get; init; }
}

public class Blog : Entity
{
    public string Title { get; set; } = string.Empty;

    public string Description { get; set; } = string.Empty;

    public Author? Author { get; set; }

    public ICollection<Post> Posts { get; set; } = new List<Post>();
}

public class Author : Entity
{
    public string Name { get; set; } = string.Empty;

    public string Description { get; set; } = string.Empty;

    public ICollection<Blog> Blogs { get; set; } = new List<Blog>();
}

public class Post : Entity
{
    public string Title { get; set; } = string.Empty;

    public string Content { get; set; } = string.Empty;

    public DateTimeOffset CreatedAt { get; set; }

    public Blog? Blog { get; set; }
}
```

Classes que l'on utilisera dans le contexte Entity Framework

Notre API proposera une route permettant derécupérer une liste d'informations sur tous les blogs : leur nom, leur description, le nom de leur auteur, ainsi que la date de leur dernière activité, qui correspondra à la date de publication du dernier article.

Nous formulons ensuite ces informations en tant que DTO.

```
public record BlogOverviewDto
{
    public required Guid Id { get; init; }

    public required string Title { get; init; }

    public required string Description { get; init; }

    public required string AuthorName { get; init; }

    public required DateTimeOffset LastActivity { get; init; }
}
```

DTO contenant des informations de base sur le blog

Le Service de Blog (BlogService)

Pour effectuer la transition de DAO à DTO, une classe de service sert d'intermédiaire entre le contrôleur et le contexte d'Entity Framework (la base de données). Elle englobe toute la logique métier de l'API et est injectée dans le contrôleur ASP.

Mettons en place une implémentation simple d'un `BlogService` :

```
public interface IBlogService
{
    IEnumerable<BlogOverviewDto> GetAll();
}

internal class BlogService : IBlogService
{
    private readonly AppDbContext context;

    public BlogService(AppDbContext context)
    {
        this.context = context;
    }

    public IEnumerable<BlogOverviewDto> GetAll()
    {
        List<Blog> blogs = context.Blogs
            .Include(b => b.Author)
            .Include(b => b.Posts.OrderByDescending(p => p.CreatedAt).Take(1))
            .ToList();

        return blogs.Select(b => new BlogOverviewDto
        {
            Id = b.Id,
            Title = b.Title,
            Description = b.Description,
            AuthorName = b.Author.Name,
            LastActivity = b.Posts.Single().CreatedAt
        });
    }
}
```

Pour simplifier l'exemple, on supposera qu'un blog a toujours au moins un post.

On récupère la liste de nos blogs, en ordonnant les posts par ordre de création et on crée ensuite notre DTO.

Nous pourrions certes créer une classe “BlogMapper” dotée d'une méthode `ToBlogOverviewDto` (et des tests unitaires associés), mais cela serait à la fois chronophage et source potentielle d'erreurs. En effet, toute évolution des objets DAO ou DTO nécessiterait une mise à jour manuelle du processus de mappage.

Nous préférerons donc utiliser le package AutoMapper, qui nous permet de supprimer cette redondance et de simplifier grandement la conversion entre DAO et DTO.

Présentation d'AutoMapper

AutoMapper est une bibliothèque conçue pour simplifier le mappage d'objet à objet, tout en minimisant la configuration requise. Si une propriété porte le même nom dans l'objet source et dans l'objet de destination, le mappage se fait automatiquement entre ces deux propriétés.

Par exemple, la propriété `Blog.Title` sera assignée à la propriété `BlogOverviewDto.Title` automatiquement. Le "Flattening" de propriété est également supporté : `Blog.Author.Name` sera donc mappé vers `BlogOverviewDto.AuthorName`.

Pour configurer la bibliothèque, il suffit de créer une classe héritant de `Profile` dans laquelle mettre notre configuration :

```
public class MappingProfile : Profile
{
    public MappingProfile()
    {
        CreateMap<Blog, BlogOverviewDto>(MemberList.Destination)
            .ForMember(
                blogDto => blogDto.LastActivity,
                option => option.MapFrom(blog => blog.Posts
                                                     .Single()
                                                     .CreatedAt));
    }
}
```

Classe de configuration des mappings

`CreateMap` est utilisé pour configurer quels objets peuvent être mappés. Ici, on a juste besoin de pouvoir transformer un objet de type Blog en objet de type `BlogOverviewDto`.

Toutes les propriétés du DTO sont mappées implicitement, à l'exception de : `LastActivity`, qui ne correspond à aucune propriété côté DAO. Il faut donc préciser à AutoMapper comment retrouver la valeur pour cette propriété dans l'objet source.

La méthode `ForMember` permet d'indiquer comment récupérer cette propriété en manuel. Il est possible de définir une opération à réaliser sur l'objet source via `option.MapFrom`, de retourner une valeur statique, d'utiliser une classe de conversion via `option.ConvertUsing`, ou bien d'ignorer la propriété avec `option.Ignore`.

Il est également possible de customiser les conventions utilisées pour le matching implicite, par Map ou par Profile : Convention de nommage des propriétés, remplacement de mots, filtres...

AutoMapper révolutionne la transformation d'objets.

AutoMapper s'intègre parfaitement à l'injection de dépendance d'ASP.NET Core, via la méthode `AddAutoMapper`, ce qui nous permettra d'injecter l'interface `IMapper` dans le `BlogService`.

```
services.AddAutoMapper(configuration =>
    configuration.AddProfile<MappingProfile>());
```

On peut également facilement vérifier que notre mapping est correct et que l'on a oublié aucune propriété, au démarrage de l'application ou dans un test unitaire par exemple, grâce à la méthode `AssertConfigurationIsValid`, fournie par la bibliothèque :

```
public void MappingProfile_ShouldHaveValidConfiguration()
{
    var configuration = new MapperConfiguration(cfg =>
        cfg.AddProfile<MappingProfile>());

    configuration.AssertConfigurationIsValid();
}
```

Si, par exemple, la propriété `LastActivity` n'était pas correctement configurée, `AssertConfigurationIsValid` lèverait une exception, conduisant à l'échec du test.

```
AutoMapper.AutoMapperConfigurationException :
Unmapped members were found. Review the types and members below.

...

Blog -> BlogOverviewDto (Destination member list)

Unmapped properties:
LastActivity
```

Erreur levée par AssertConfigurationIsValid

Le paramètre passé à `CreateMap` permet de spécifier quelles propriétés seront validées :

- **`MemberList.Destination` :** Valide que toutes les propriétés de l'objet de destination sont mappées, c'est le comportement par défaut.
- **`MemberList.Source` :** Valide que toutes les propriétés de l'objet source sont mappées.
- **`MemberList.None` :** Pas de validation.

Si l'on voulait également transformer un `BlogOverviewDto` en `Blog`, on pourrait ajouter un ReverseMap à notre configuration.
Attention cependant : `ReverseMap` ne valide pas le mapping ! Pour valider la configuration dans les deux sens, il serait nécessaire d'ajouter également `ValidateMemberList` et préciser si l'on veut valider les membres source ou destination.

```
CreateMap<Blog, BlogOverviewDto>(MemberList.Destination)
    .ForMember(
        blogDto => blogDto.LastActivity,
        option => option.MapFrom(blog => blog.Posts
                                            .Single()
                                            .CreatedAt))
    .ReverseMap()
    .ValidateMemberList(MemberList.Source);
```

Configuration avec ReverseMap

⚠ Cette configuration ne passera pas la validation : On n'a pas précisé comment la propriété LastActivity du DTO devait être mappée dans l'objet Blog.

Transformer nos objets

Mettons maintenant à jour le `BlogService` pour utiliser notre mapping :

```
internal class BlogService : IBlogService
{
    private readonly AppDbContext context;
    private readonly IMapper mapper;

    public BlogService(AppDbContext context, IMapper mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

    public IEnumerable<BlogOverviewDto> GetAll()
    {
        List<Blog> blogs = context.Blogs
            .Include(b => b.Author)
            .Include(b => b.Posts.OrderByDescending(p => p.CreatedAt).Take(1))
            .ToList();

        return blogs.Select(mapper.Map<BlogOverviewDto>);
    }
}
```

On a un code beaucoup plus lisible, et surtout la logique de mapping est sortie du service.

Intégration avec les ORM

Jetons un œil à la requête effectuée à notre base de données :

```
SELECT
    [b].[id], [b].[AuthorId], [b].[Description], [b].[Title],
    [a].[id], [a].[Description], [a].[Name],
    [t0].[id], [t0].[BlogId], [t0].[Content], [t0].[CreatedAt], [t0].[Title]
FROM
    [Blogs] AS [b]
INNER JOIN
    [Authors] AS [a] ON [b].[AuthorId] = [a].[id]
LEFT JOIN (
    SELECT
        [t].[id], [t].[BlogId], [t].[Content], [t].[CreatedAt], [t].[Title]
    FROM (
        SELECT
            [p].[id], [p].[BlogId], [p].[Content], [p].[CreatedAt], [p].[Title],
            ROW_NUMBER() OVER(PARTITION BY [p].[BlogId] ORDER BY [p].[CreatedAt] DESC) AS [row]
        FROM
            [Posts] AS [p]
    ) AS [t]
    WHERE
        [t].[row] <= 1
) AS [t0] ON [b].[id] = [t0].[BlogId]
ORDER BY
    [b].[id], [a].[id], [t0].[BlogId], [t0].[CreatedAt] DESC
```

On récupère beaucoup plus de données que nécessaire, qui ne seront pas utilisées puisque pas présentes dans notre DTO.

Pour ne récupérer que ce dont on a besoin, on peut effectuer un `Select` pour choisir quelles propriétés demander dans la requête :

```
public IEnumerable<BlogOverviewDto> GetAll()
{
    return context.Blogs
        .Select(b => new BlogOverviewDto
        {
            Id = b.Id,
            Title = b.Title,
            Description = b.Description,
            AuthorName = b.Author.Name,
            LastActivity = b.Posts
                .OrderByDescending(p => p.CreatedAt)
                .First()
                .CreatedAt
        });
}
```

Ce qui nous donne la requête SQL suivante :

```
SELECT
    [b].[id] AS [Id], [b].[Title], [b].[Description],
    [a].[Name] AS [AuthorName], (
        SELECT TOP(1) [p].[CreatedAt]
        FROM [Posts] AS [p]
        WHERE [b].[id] = [p].[BlogId]
        ORDER BY [p].[CreatedAt] DESC) AS [LastActivity]
FROM
    [Blogs] AS [b]
INNER JOIN
    [Authors] AS [a] ON [b].[AuthorId] = [a].[id]
```

C'est bien mieux, par contre, on se retrouve à nouveau à faire du mapping manuel, ce qui est précisément ce que l'on voulait éviter en utilisant AutoMapper.

Heureusement pour nous, celui-ci va une fois de plus nous simplifier la vie, grâce à la méthode `ProjectTo` cette fois.

Projection en base de données

AutoMapper peut très facilement s'intégrer avec Entity Framework (ou n'importe quel autre ORM d'ailleurs), grâce à des extensions de l'interface `IQueryable`, notamment la méthode `ProjectTo`.

Contrairement à la méthode `Map` qui effectue la transformation d'objet dans la mémoire, `ProjectTo` va construire une instruction `SELECT` à partir d'un `IQueryable`, le mapping va donc se faire directement côté base de données.

Mettons à jour la configuration pour y intégrer la même logique que dans notre `Select` précédent :

```
CreateMap<Blog, BlogOverviewDto>(MemberList.Destination)
    .ForMember(
        blogDto => blogDto.LastActivity,
        option => option.MapFrom(blog => blog.Posts
                                            .OrderByDescending(p => p.CreatedAt)
                                            .First()
                                            .CreatedAt));
```

Il suffit ensuite d'appeler `ProjectTo`, en lui passant en paramètre la configuration du mapper :

```
public IEnumerable<BlogOverviewDto> GetAll()
{
    return context.Blogs.ProjectTo<BlogOverviewDto>(mapper.ConfigurationProvider);
}
```

On peut difficilement faire plus simple.
Si on vérifie notre requête SQL, on obtient exactement la même chose qu'avec notre `Select` manuel.

On a donc à la fois une requête SQL simplifiée, et la validation de toute notre logique de mapping via AutoMapper.

> ⚠ Bien sûr, comme la transformation de l'objet se fait en base de données, cela limite la configuration que l'on peut mettre en place : Par exemple, impossible d'utiliser des classes de conversion customisées ou du mapping conditionnel.

Mesures de performances

Pour essayer de mesurer la différence entre les différentes solutions, on peut réaliser un benchmark à l'aide de la bibliothèque BenchmarkDotNet :

Méthode de mapping	Temps d'execution	Mémoire allouée
En mémoire, manuel	118.07 ms	8001.74 KB
En mémoire, avec AutoMapper	133.85 ms	8257.86 KB
En base de données, manuel	290.3 ms	758.67 KB
En base de données, avec AutoMapper	285.2 ms	757.82 KB

Test réalisé sur une base de données avec 1000 blogs, chacun avec 100 posts

On peut déjà voir que l'impact de l'utilisation d'AutoMapper est négligeable Le mapping côté base de données est légèrement plus lent dans notre cas, dû à des optimisations faites par défaut par Entity Framework dans sa requête initiale (via lepartitionnement et la jointure de la table `Blogs`) qu'on ne retrouve pas dans la requête simplifiée.

Pour essayer de comparer à requête SQL équivalente, modifions la requête utilisée par Entity Framework lors du mapping en mémoire :

```
SELECT
    [b].[id], [b].[AuthorId], [b].[Description], [b].[Title],
    [a].[id], [a].[Description], [a].[Name],
    [p].[id], [p].[BlogId], [p].[Content], [p].[CreatedAt], [p].[Title]
FROM
    [Blogs] AS [b]
INNER JOIN
    [Authors] AS [a] ON [b].[AuthorId] = [a].[id]
LEFT JOIN [Posts] AS [p] ON [p].[id] = (
    SELECT TOP 1 [p2].[id]
    FROM [Posts] AS [p2]
    WHERE [p2].[BlogId] = [b].[id]
    ORDER BY [p2].[CreatedAt] DESC
)
```

Ce qui nous donne :

Méthode de mapping	Temps d'execution	Mémoire allouée
En mémoire	280.1 ms	10080.33 KB
En base de données	265.5 ms	757.45 KB

Non seulement le mapping en base de données est légèrement plus rapide, mais surtout, dans tous les cas, nous pouvons consommer jusqu'à 10 fois moins de mémoire, ce qui peut être intéressant si on a à mapper des collections de taille importante.

Si la configuration d'un mapping est compatible avec une execution en base de données, utiliser la projection peut donc être un bon moyen de facilement optimiser ses requêtes.

Le mot de la fin

AutoMapper est extrêmement pratique pour éviter la mise en place de logiques de mapping répétitives. Néanmoins, comme l'énonce son créateur Jimmy Bogard dans son post AutoMapper's Design Philosophy :

AutoMapper works because it enforces a convention. It assumes that your destination types are a subset of the source type. It assumes that everything on your destination type is meant to be mapped. It assumes that the destination member names follow the exact name of the source type. It assumes that you want to flatten complex models into simple ones.

AutoMapper est parfaitement adapté à un use case en particulier : pour transformer des objets complexes vers une représentation plus simple. Comme on l'a vu avec la propriété `LastActivity`, dès que les objets à mapper ne correspondent pas exactement, on se retrouve à mettre en place de la configuration manuelle plus ou moins complexe. Et plus on a besoin d'adapter manuellement sa configuration, plus on perd le bénéfice d'avoir un mapping automatique.

Il est donc important de déterminer si la mise en place d'AutoMapper est pertinente selon les besoins de l'application.

If you find yourself hating a tool, it's important to ask - for what problems was this tool designed to solve? And if those problems are different than yours, perhaps that tool isn't a good fit.

Au-delà du "with": les gestionnaires de contexte

Par Alexandre MOEVI.

Photo by Maarten van den Heuvel / Unsplash

Plongeons dans les gestionnaires de contexte en Python. L'instruction with n'aura plus de secret pour vous !

Si vous avez déjà codé en Python, il est fort probable que vous ayez utilisé le mot-clé `with` de cette manière :

```
with open("sfeir.txt") as file:
    data = file.read()
    print(data)
```

Cependant, le fonctionnement précis de l'instruction `with` reste souvent méconnu. En jetant un regard en coulisses et en explorant les gestionnaires de contexte sous-jacents, nous allons découvrir les mécanismes qui donnent vie à cette instruction.

Contexte ?

L'instruction `with` offre une méthode simple et puissante pour gérer les ressources à l'aide de gestionnaires de contexte. Cette approche suit le patron de conception classique `try... except... finally` et permet d'encapsuler l'exécution d'un bloc de code de manière propre et sécurisée.

Un gestionnaire de contexte (ou *context manager*) gère donc l'acquisition et la libération de ressources dans un bloc de code. Il a pour but de garantir que les ressources sont correctement initialisées et nettoyées, même en cas d'erreur ou d'exception.

Un context manager doit implémenter deux méthodes spéciales : `__enter__()` et `__exit__()`. La méthode `__enter__()` est appelée au début du bloc `with` et est responsable de l'initialisation des ressources. Elle renvoie généralement l'objet qui sera associé au gestionnaire de contexte.

La méthode `__exit__()` est appelée à la fin du bloc `with` et est responsable de la libération des ressources.

Avec ce premier gestionnaire basique, on voit clairement l'ordre d'exécution.

```
class MyContextManager:
    def __enter__(self):
        print("Initialisation des ressources")
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        print("Nettoyage des ressources")

with MyContextManager() as cm:
    print("Exécution du bloc de code")
```

```
$ python contexte.py
Initialisation des ressources
Exécution du bloc de code
Nettoyage des ressources
```

Ainsi, on peut s'amuser à re-coder un context manager qui va lire ou écrire dans des fichiers.

On utilse la fonction `open()` qui nous renvoie un flux de données[1]. Le flux est ouvert dans la méthode `__enter__()` et on n'oublie pas de le fermer dans la fonction `__exit__()`.

```
class Fichier:
    def __init__(self, nom, mode):
        self.nom = nom
        self.mode = mode
    def __enter__(self):
        self.fichier = open(self.nom, self.mode)
        return self.fichier
    def __exit__(self, exc_type, exc_value, traceback):
        self.fichier.close()


with Fichier("sfeir.txt", 'w') as f:
    f.write("sfeir.dev")
```

La classe, non ?

Python fournit également le module `contextlib` qui propose un décorateur `@contextmanager` afin de créer des gestionnaires de contexte de manière plus concise, sans avoir à définir une classe complète. Si on remplace notre classe `Fichier`, ça donne ceci :

[1] Si vous avez bien suivi, `open` retourne un objet qui est un flux, mais qui est aussi un context manager qu'on peut utiliser avec `with`. Pour en savoir plus, voici les liens vers la documentation de la fonction open et de l'objet IOBase.

```python
from contextlib import contextmanager

@contextmanager
def fichier(nom, mode):
    try:
        f = open(nom, mode)
        yield f
    finally:
        f.close()

with fichier("sfeir.txt", 'w') as f:
    f.write("sfeir.dev")
```

Avec le décorateur, on transforme la fonction `fichier()` en un gestionnaire de contexte. La fonction utilise le mot-clé `yield` pour indiquer le point d'entrée et de sortie du contexte. Autrement dit, on prépare les ressources nécessaires pour le contexte avant `yield` puis on les détruit après.

Dans notre code, on utilise `yield` pour retourner le flux qui sera utilisé à l'intérieur du bloc `with`. Une fois le bloc terminé, la partie `finally` est exécutée pour fermer le fichier.

Les context managers en action

On retrouve les gestionnaires de contexte aussi bien dans la librairie standard de Python que dans les modules populaires. Voici quelques exemples supplémentaires de gestionnaires de contexte dans différents domaines.

```
# Gestion d'une connexion vers une base données
import sqlite3

with sqlite3.connect("database.db") as conn:
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM table")
    results = cursor.fetchall()
    print(results)



# Gestion d'une session dans le module requests
# Une session conserve les paramètres et les cookies entre plusieurs requêtes
import requests

with requests.Session() as session:
    # Effectuer des requêtes HTTP ici
    response = session.get("https://www.sfeir.dev")
    print(response.status_code)



# Gestion d'une connexion vers un Redis
import redis

with redis.Redis(host='localhost', port=6379) as r:
    r.set('cle', 'valeur')
    valeur = r.get('cle')
    print(valeur)
```

```python
# Gestion de threads concurrents
# Ici, 10 threads doivent incrémenter la même variable
# Sans le lock, les threads peuvent accéder simultanément à la variable partagée,
# ce qui peut entraîner des incohérences et des erreurs de valeur.
# Le lock permet d'assurer qu'un seul thread à la fois peut modifier la variable,
# garantissant ainsi la cohérence des données.
import threading

lock = threading.Lock()
counter = 0

def thread_function():
    global counter
    thread_name = threading.current_thread().name
    for _ in range(10):
        with lock:
            counter += 1
            print(f"{thread_name} incrémente, valeur du compteur : {counter}")

threads = [threading.Thread(target=thread_function, name=f"Thread {i}") for i in range(1, 11)]

for thread in threads:
    thread.start()

for thread in threads:
    thread.join()
```

Niveau supérieur

Pour aller plus loin, les gestionnaires peuvent se souvenir de l'état précédent et adapter leur comportement en fonction du contexte. Cela permet de créer des contextes imbriqués où certaines actions sont effectuées à l'entrée et à la sortie de chaque niveau de contexte. On parle alors de gestionnaire réentrant. Voici un exemple de ce type de context manager :

```
class Indentation:
    def __init__(self):
        self.niveau = 0

    def __enter__(self):
        self.niveau += 1
        return self

    def __exit__(self, exc_type, exc_val, traceback):
        self.niveau -= 1

    def imprimer(self, texte):
        print(f"{'    ' * self.niveau}{texte}")


with Indentation() as indent:
    indent.imprimer('Niveau 1')
    with indent:
        indent.imprimer('Niveau 2')
        with indent:
            indent.imprimer('Niveau 3')
    indent.imprimer('Encore niveau 1')
```

```
$ python contexte.py
    Niveau 1
        Niveau 2
            Niveau 3
    Encore niveau 1
```

Dans cet exemple, `Indentation` maintient un état `niveau` qui est incrémenté à chaque niveau de contexte imbriqué et décrémenté lors de la sortie de chaque niveau. Cela permet d'adapter le comportement du manager réentrant en fonction du niveau de contexte actuel. En sortie, on voit la structure qui reflète la profondeur du contexte grâce à l'indentation.

Pour les curieux

Je vous renvoie vers la documentation officielle du module `contextlib`. Cette page revient sur le décorateur que l'on a rapidement évoqué mais parle aussi des gestionnaires de contexte asynchrones, des gestionnaires réutilisables comme le `Lock` ou encore du gestionnaire `ExitStack`.

Le gestionnaire de contexte assure une gestion impeccable des ressources, même en cas d'erreur.

Collaboration et innovation

Dans le développement Back-End, chaque ligne de code est une étape vers l'innovation et la créativité.

[sfeir]

Au travers ces deux articles d'experts, cette section discute de l'importance de la collaboration et de l'innovation dans le développement Back-End, et comment elles peuvent être intégrées dans les pratiques quotidiennes.

- **Devenez accro aux crochets git.** Cet article guide sur l'utilisation des hooks Git pour automatiser les tâches de développement et améliorer la qualité des projets. Il explique comment ces scripts personnalisables s'exécutent à différentes étapes du cycle de vie Git, et offre des exemples pratiques pour leur mise en œuvre en utilisant divers langages de programmation.

- **Ouverture des inscriptions à l'Hacktoberfest !** L'Hacktoberfest est un événement annuel qui incite la communauté des développeurs à contribuer à des projets open source. Cet article souligne l'importance de cette participation pour l'apprentissage, le développement de compétences et la satisfaction personnelle dans le respect de la philosophie open source.

Devenez accro aux crochets git

Par Alexandre MOEVI.

Photo by RealToughCandy.com / Pexels

Apprenez à créer des hooks git pour automatiser vos tâches de développement et améliorer la qualité de vos projets.

Si vous travaillez avec `Git`, vous avez certainement remarqué le dossier .git à la racine de vos projets.

Mais connaissiez-vous l'existence du sous-dossier `hooks` ?

```
$ ls -l .git/hooks
total 60
-rwxr-xr-x 1 amoevi amoevi  478 avril 23 11:20 applypatch-msg.sample
-rwxr-xr-x 1 amoevi amoevi  896 avril 23 11:20 commit-msg.sample
-rwxr-xr-x 1 amoevi amoevi 4655 avril 23 11:20 fsmonitor-watchman.sample
-rwxr-xr-x 1 amoevi amoevi  189 avril 23 11:20 post-update.sample
-rwxr-xr-x 1 amoevi amoevi  424 avril 23 11:20 pre-applypatch.sample
-rwxr-xr-x 1 amoevi amoevi 1643 avril 23 11:20 pre-commit.sample
-rwxr-xr-x 1 amoevi amoevi  416 avril 23 11:20 pre-merge-commit.sample
-rwxr-xr-x 1 amoevi amoevi 1492 avril 23 11:20 prepare-commit-msg.sample
-rwxr-xr-x 1 amoevi amoevi 1374 avril 23 11:20 pre-push.sample
-rwxr-xr-x 1 amoevi amoevi 4898 avril 23 11:20 pre-rebase.sample
-rwxr-xr-x 1 amoevi amoevi  544 avril 23 11:20 pre-receive.sample
-rwxr-xr-x 1 amoevi amoevi 2783 avril 23 11:20 push-to-checkout.sample
-rwxr-xr-x 1 amoevi amoevi 3650 avril 23 11:20 update.sample
```

Les hooks (ou crochets) sont des scripts personnalisables qui s'exécutent automatiquement à différentes étapes du cycle de vie Git. Nous allons découvrir leur potentiel et comment les utiliser afin d'automatiser certaines tâches pendant le développement.

La théorie

Un hook Git est donc un script qui s'exécute avant ou après des événements dans un dépôt Git. Il en existe deux types :

- **Les hooks coté client:** ils sont déclenchés localement par des opérations telles qu'un `git commit` ou un `git merge`.

- **Les hooks coté serveur :** ils sont exécutés par le serveur hébergeant le code et sont déclenchés lorsqu'il reçoit du code à la suite d'un `git push`.

Les crochets côté client sont locaux et ne sont pas copiés. Si vous comptez utiliser les scripts pour faire respecter les mêmes règles pour tous les développeurs travaillant sur un projet, il est recommandé d'utiliser des hooks serveur.

Par défaut, Git offre des exemples de crochets à la création d'un dépôt. Ils se trouvent dans le dossier `.git/hooks`, qu'on a vu en introduction.

Les hooks `pre`, tels que `pre-commit` ou `pre-push`, peuvent annuler l'action Git correspondante s'ils se terminent avec un code de sortie non nul. C'est donc avec ces crochets qu'il est intéressant de mettre en place des vérifications.

La pratique

Pour activer un hook, enlevez simplement le suffixe `.sample`, et il sera exécuté au moment de l'événement correspondant dans Git. Si vous partez de vos propres scripts, n'oubliez pas de vérifier les noms des fichiers correspondent aux actions Git et de contrôler les permissions d'exécution.

Les fichiers dans le dossier `.git/hooks` sont des scripts shell avec un peu de Perl. Mais vous pouvez implémenter les crochets avec votre langage favori, tant que les fichiers sont nommés correctement et exécutables. Par exemple, on peut implémenter le hook `pre-commit` avec du Javascript.

```
#!/usr/bin/env node

const fs = require('fs');

if (!fs.existsSync('README.md')) {
  console.error('Erreur : README.md n\'existe pas.');
  process.exit(1);
}

process.exit(0);
```

.git/hooks/pre-commit

Le script `pre-commit` s'exécute à chaque fois que vous lancez git commit, avant que Git vous demande d'entrer un message de commit. Dans notre cas, le commit est annulé car il manque le README dans le dossier.

```
$ ls
src     dist    lib
$ git commit
Erreur : README.md n'existe pas.
```

Autre vérification possible, on peut lancer une série de tests avant de réaliser un commit.

Si les tests fonctionnent, tout va bien. Sinon, l'opération est annulée.

```
#!/bin/sh

echo "Exécution des tests Java..."

# Exécutez les tests avec Maven
mvn test

# Vérifiez le code de sortie de Maven
if [ $? -ne 0 ]; then
  echo "Les tests ont échoué. Annulation du commit."
  exit 1
fi

echo "Tous les tests ont réussi. Continuation du commit."
exit 0
```

.git/hooks/pre-commit

Contrairement au `pre-commit`, certains hooks reçoivent des paramètres qui sont utilisables lors de leur exécution.

C'est le cas du `commit-msg`, qui reçoit le chemin du fichier contenant le message de commit. On peut ainsi utiliser cette valeur pour vérifier que le message de commit suit les règles du projet.

```python
#!/usr/bin/env python3

import sys
import re

commit_msg_filepath = sys.argv[1]

with open(commit_msg_filepath, 'r') as file:
    commit_msg = file.read()

pattern = re.compile(r'^SFEIR-\d+: .+')

if not pattern.match(commit_msg):
    print("Le message de commit ne suit pas le format attendu (SFEIR-123: Description)")
    sys.exit(1)
```

.git/hooks/commit-msg

On peut observer l'effectivité du crochet avec les codes de sortie de la commande `git commit` .

```
$ git add sfeir.txt
$ git commit -m "Adding new fil
Le message de commit ne suit pa le format attendu (SFEIR-123: Description)
$ echo $?
1
$ git commit -m "SFEIR-1: Addin new file"
[githooks 0ff1ce] SFEIR-1: Addi new file
$ echo $?
0
```

Enfin, nous avons évoqué les hooks côté serveur et le fait que des hooks puissent se déclencher après l'évènement. Pour illustrer cela, on peut imaginer un crochet `post-receive` qui envoie un message sur un canal Slack dès que la branche `main` reçoit de nouveaux changements.

```
#!/bin/bash

# URL du webhook Slack
WEBHOOK_URL="URL_WEBHOOK_SLACK"

# Lire les informations du push
while read oldrev newrev refname; do
  # Vérifiez si le push est sur la branche main
  if [ "$refname" = "refs/heads/main" ]; then
    # Construire le message JSON
    JSON_TEXT="{\"text\":\"Un nouveau push a été effectué sur la branche main: $newrev\"}"

    # Envoyer la notification Slack
    curl -X POST --data-urlencode "payload=$JSON_TEXT" $WEBHOOK_URL
  fi
done
```

.git/hooks/post-receive

La documentation

Il existe d'autres hooks que nous n'avons pas abordés, comme le `prepare-commit-msg` qui permet de réécrire le message par défaut dans les commits ou le `post-checkout` qui est appelé après un `git checkout` ou `git switch`.

La commande `man githooks` décrit les crochets ainsi que leur utilisation. Il existe aussi une version web de cette page du manuel.

Sur ce même site, vous pouvez également trouver un exemple de politique git, qui implémente des hooks côté serveur et client en Ruby.

Ouverture des inscriptions à l'Hacktoberfest !

10 ans de l'Hacktoberfest !

Par Erwan LE TUTOUR.

Cette année marque la dixième édition de l'hacktoberfest, célébrons l'open source avec des contributions ! #opensource #hacktoberfest

Hacktoberfest ? Mais ce n'est pas plutôt l'Oktoberfest, la célèbre fête de la bière que l'on célèbre en octobre ? Si, mais à la place de la bière, nous préférons célébrer l'open source !

Qu'est ce que l'Hacktoberfest ?

L'Hacktoberfest comme son homonyme consacré à la bière, est un évenement annuel qui encourage l'ensemble de la communauté dev à contribuer à des projets open source. Une grande partie des projets sur lesquels nous intervenons au quotidien chez nos clients, ou bien nos projets personels reposent sur des projets open source maintenus par des personnes passionnées qui souvent n'ont pas les moyens suffisant pour faire autre chose que maintenir le projet en vie. L'Hacktoberfest vise à mettre ses projets en avant afin de les aider.

Comment y participer ?

Pour participer à l'Hacktoberfest rien de plus simple, il suffit de se rendre à l'adresse suivante https://hacktoberfest.com/participation/ et de s'inscrire entre le **26 septembre** et le **31 octobre** avec son Github ou Gitlab.

Une fois cela fait, dès que le compte à rebours arrive à zéro, vous pourrez commener à envoyer vos **merge request**.

compte à rebours Hacktoberfest

Envoyer des merges request ? Mais comment on peut savoir sur quels projets contribuer ? Là encore, c'est très simple, il suffit d'aller sur Github ou Gitlab et trouver des projets qui ont le tag #Hacktoberfest.

Ensuite pour que vos merge request soit comptabilisées il faut que les personnes en charge des projets sur lesquels vous les avez faites les acceptent et qu'il n'y ai pas de retour arrière dessus pendant une période de temps donnée (7 jours).

Dès que vous avez 4 merge request acceptées entre le 1er octobre et le 31 octobre, vous aurez validé le challenge de contribution à l'open source qu'est l'Hacktoberfest.

Comment contribuer à un projet open source ?

Choisir un rôle

Au sein d'un projet open source, il existe divers rôles et responsabilités, chaque participant pouvant avoir une place spécifique en fonction de ses compétences, de son niveau d'implication et de ses intérêts. Voici quelques-uns des principaux rôles au sein d'un projet open source :

— Mainteneur de Projet
- Responsable de la gestion globale du projet.
- Prend des décisions importantes sur l'orientation, la stratégie et les grandes fonctionnalités du projet.
- Gère le code source, les versions, la coordination de la communauté et les problèmes critiques.
- Effectue des revues de code, fusionne les contributions et assure la qualité du code.

— Contributeur
- Soumet des contributions au projet, telles que des correctifs de bogues, des améliorations de fonctionnalités ou de la documentation.
- Suit les directives de contribution du projet, notamment les normes de codage.
- Participe aux discussions sur les problèmes et les demandes de fonctionnalités.
- Peut effectuer des revues de code pour les contributions des autres.

— Testeur/QA (Assurance Qualité)
- Teste le logiciel pour identifier et signaler les bugs.
- Fournit des rapports de bogues détaillés.
- Peut aider à créer des jeux de tests et à automatiser les tests.

— Documentaliste
- Rédige et met à jour la documentation du projet.
- Crée des guides d'utilisation, des tutoriels et des exemples.
- Veille à ce que la documentation soit précise et à jour.

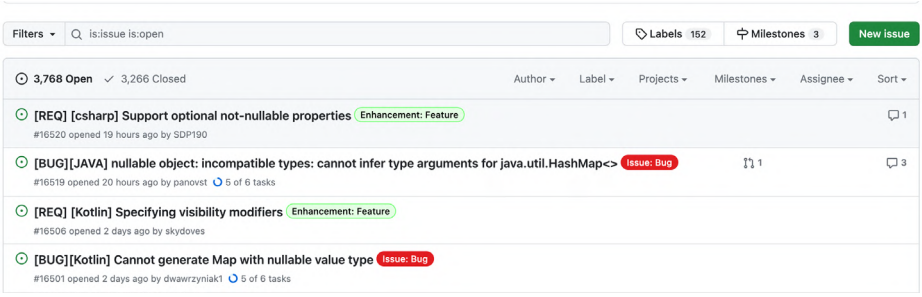
Chaque position au sein d'un projet open source revêt une importance capitale, car elle apporte sa pierre à l'édifice de la réussite et à l'expansion du projet. Les projets open source sont généralement le fruit de la coopération d'individus aux compétences et intérêts variés. Les contributeurs peuvent endosser simultanément plusieurs de ces rôles, en fonction de leurs compétences et de leur disponibilité.

Choisir un projet

À l'heure où j'écris ces lignes, il y'a actuellement 125511 projets publics sur Github avec le #Hactoberfest ce qui represente la partie émergé de l'iceberg, autrement dit vous avez le choix dans la matière, il ne vous reste plus qu'à filtrer par langage pour trouver ce que vous voulez faire :

- travailler avec votre techno de prédilection
- vous formez sur une nouvelle techno

Une fois le projet choisit vous n'avez qu'à en lire la documentation (généralement le fichier README.md), le guide de contribution, et de regarder la liste des issues ouvertes (plus ou moins nombreuses et complexe en fonction du projet).



Et ensuite ? Ensuite c'est comme dans les projets sur lesquels vous intervenez au quotidien : du test, du debug, des essais concluant ou non, et dès que vous avez atteind le résultat voulu, vous transformez l'essai avec une merge / pull request.

Pourquoi contribuer à l'open source ?

1. **Respect de la philosophie open source :** Si vous croyez en la philosophie de l'open source, qui promeut la transparence, la collaboration et le partage des connaissances, contribuer à des projets open source est une manière de soutenir ces valeurs.

2. **Construction de réseaux et de relations :** En contribuant à des projets open source, vous avez l'opportunité de collaborer avec des personnes du monde entier. Cela peut vous aider à élargir votre réseau et vous permet d'établir des relations avec des experts du domaine.

3. **Apprentissage et développement des compétences :** Participer à des projets open source est un excellent moyen d'apprendre de vous former sur de nouvelles techno et de perfectionner celles que vous maitrisez déjà.

4. **Satisfaction personnelle :** On en parle pas assez, mais le fait de savoir que son travail est utilisé par d'autre et qu'il peut bénéficier à avoir un impact positif sur la communauté open source permet d'avoir une grande satisfaction personnelle.

L'Hacktoberfest, c'est l'esprit de l'open source au service de la communauté tech.

La vision de SFEIR pour l'avenir du développement Back-End

Marier l'innovation technologique, l'excellence technique et l'agilité opérationnelle, pour transformer chaque défi en succès.

Chez SFEIR, nous sommes animés par la conviction que l'avenir du développement Back-End réside dans une harmonie parfaite entre innovation technologique, excellence technique, et agilité opérationnelle. Guidés par des visionnaires comme Didier GIRARD, nous explorons sans cesse de nouvelles avenues pour repousser les frontières du possible, tout en restant fermement engagés envers la qualité et la fiabilité.

Notre vision pour l'avenir se profile comme un paysage technologique où les solutions Back-End ne sont pas simplement les fondations des applications, mais aussi les catalyseurs d'innovation et de transformation numérique. Nous envisageons un avenir où les technologies Back-End sont plus flexibles, plus sécurisées, et parfaitement adaptées aux besoins changeants des entreprises et des utilisateurs finaux. Avec un œil toujours tourné vers les dernières avancées, qu'il s'agisse d'adopter de nouvelles architectures, d'explorer l'informatique quantique, ou d'intégrer l'intelligence artificielle, SFEIR s'engage à rester à l'avant-garde du développement Back-End.

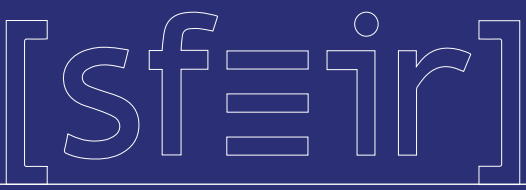
Notre engagement dépasse la simple adaptation aux tendances actuelles; nous aspirons à être les créateurs et les innovateurs de ces tendances.

Cet engagement se manifeste dans notre approche rigoureuse de la formation continue de nos équipes, notre investissement constant dans la recherche et le développement, et notre collaboration étroite avec nos partenaires et clients pour co-créer des solutions uniques et personnalisées.

Nous comprenons que l'innovation n'est pas une fin en soi, mais un moyen de fournir une valeur ajoutée réelle et mesurable à nos clients. En tant que partenaire stratégique, SFEIR est dédié à transformer les défis technologiques de ses clients en succès commerciaux, en s'appuyant sur notre expertise en développement Back-End pour construire des solutions qui répondent non seulement aux besoins actuels mais anticipent également les exigences futures.

En plaçant ces valeurs au cœur de tout ce que nous faisons, nous continuons à façonner l'avenir du développement technologique, propulsant nos clients et l'ensemble de l'industrie vers de nouveaux sommets d'innovation et de succès.

Chez SFEIR, nous ne suivons pas les tendances, nous les créons pour propulser l'innovation et le succès.



Allez plus loin et développez vos compétences technologiques avec SFEIR Institute

SFEIR Institute propose des solutions de formation complètes et officielles via des partenariats avec les principaux fournisseurs de technologies, notamment AWS, Confluent, Google Cloud, The Linux Foundation, et Microsoft Azure. SFEIR Institute développe également ses propres cours, notamment IA, Big Data, Cloud, Kubernetes, DevOps, FinOps, Développement Front-End et Back-End, IoT, Mobile, Web. Inter ou Intra.

Nos programmes de formation backend

Ils sont conçus pour répondre aux besoins des individus et des organisations souhaitant renforcer leur expertise technique en développement et opérations Back-End. Avec une gamme complète de cours couvrant Kubernetes, Docker et une série de technologies DevOps incluant Terraform, Git et GitLab, nous accélérons la montée en compétence des professionnels dans l'écosystème numérique moderne.

Formation Kubernetes

Nos formations Kubernetes couvrent deux métiers : Administrateur et Développeur. La formation Administrateur est adaptée pour les administrateurs systèmes et les professionnels DevOps, se concentrant sur la configuration, la gestion et le dépannage des clusters. La formation Développeur, d'autre part, est conçue pour les développeurs de logiciels, mettant l'accent sur le développement, le déploiement et la mise à l'échelle des applications dans des environnements Kubernetes.

Formation Docker

Docker est devenu une pierre angulaire dans le développement et le déploiement des applications modernes. Notre formation Docker démystifie la technologie de conteneurisation, guidant les participants à travers le processus de conteneurisation des applications, la gestion du cycle de vie des conteneurs et l'optimisation des flux de travail Docker pour les environnements de développement et de production.

Technologies DevOps

Dans notre gamme de cours DevOps, les participants peuvent approfondir les outils et les pratiques qui permettent aux organisations à haute vélocité. Avec des cours sur Terraform, Ansible, Git et GitLab, les apprenants maîtriseront l'infrastructure en tant que code, l'automatisation, le contrôle de version et les pipelines d'intégration continue/déploiement continu (CI/CD), les dotant des capacités pour rationaliser les cycles de développement et améliorer l'efficacité opérationnelle.

Formats de Livraison Flexibles

Comprenant les besoins et contraintes diversifiés des équipes modernes, nous proposons notre formation dans des formats en présentiel et virtuel dirigés par un instructeur. Notre formation en présentiel offre une expérience pratique et interactive, favorisant la collaboration et l'engagement direct avec nos instructeurs experts. Pour les équipes réparties sur plusieurs géographies ou celles préférant la commodité de l'apprentissage à distance, nos classes virtuelles offrent une solution flexible et efficace, utilisant des plateformes de pointe pour fournir une expérience d'apprentissage engageante et interactive.

Solutions de Formation Personnalisées

Reconnaissant que chaque organisation a des défis et objectifs uniques, nous proposons des programmes de formation personnalisés adaptés à vos besoins spécifiques. Qu'il s'agisse d'adapter le programme pour se concentrer sur des domaines d'intérêt particuliers, d'incorporer des études de cas pertinentes à votre secteur ou d'aligner le calendrier de formation avec la disponibilité de votre équipe, notre objectif est de fournir une solution de formation qui s'aligne parfaitement sur vos objectifs et maximise les résultats d'apprentissage pour votre équipe.

Plus d'informations à propos des formations Back-End proposées par SFEIR Institute : institute.sfeir.com/backend

À propos des auteurs

Yves DAUTREMAY
Développeur Agile et Back-End, SFEIR
Strasbourg

Fabian PIJCKE
Développeur Agile et Back-End, SFEIR
Luxembourg

Guillaume FOURNY
Développer Back-End Senior, SFEIR
Nantes

Alexandre MOEVI
Développeur Back-End, SFEIR Lille

Mathieu VANDENNEUCKER
Développeur Full-Stack, SFEIR Belgium

Gaëlle CHALOT
Développeuse Full-Stack, SFEIR Strasbourg

Erwan LE TUTOUR
Ingénieur d'étude et de développement, SFEIR Paris

Charlotte RYSSEN
Content Manager et Rédactrice en Chef de sfeir.dev, SFEIR Groupe

Conçu, réalisé et édité par SFEIR.

[sfeir]

contact@sfeir.com
www.sfeir.com

Paris
48, rue Jacques Dulud
92200 Neuilly-sur-Seine
+33 1 41 38 52 00

Lille
74, rue des Arts
59800 Lille
+33 3 66 72 61 32

Strasbourg
Crystal Park
1, avenue de l'Europe
67300 Schiltigheim
+33 3 88 47 04 38

Luxembourg
5, Place de la Gare
1616 Luxembourg
+352 26 54 47

Bordeaux
5, rue de Condé
33000 Bordeaux
+33 5 54 07 19 72

Nantes
Halle 6 Est
40 rue de la Tour
d'Auvergne
44200 Nantes
+33 2 55 07 42 61

Belgique
Avenue des Arts 6
1210 Bruxelles
+32(0)2 899 83 70