

DOJO HEPIA – Une plateforme d'apprentissage de programmation en ligne



Thèse de Bachelor présentée par

Alexandre Vanini

pour l'obtention du titre Bachelor of Science HES-SO en

**Ingénierie des technologies de l'information avec orientation en
Logiciels et systèmes complexes**

Septembre 2019

Professeur HES responsable

Joël Cavat

SOMMAIRE DÉTAILLÉ

Remerciements	9
Énoncé du sujet	11
Résumé	13
Table des illustrations.....	15
Index des tableaux.....	17
Table des annexes.....	17
Table des listing	19
Liste des acronymes	19
Introduction	21
DojoHepia.....	22
Méthodologie de travail.....	23
User Story Mapping.....	23
Avant-propos	24
Vocabulaire	24
Figures.....	24
Listing	24
Chapitre 1 : Analyse fonctionnelle.....	25
1.1 Analyse du marché et des besoins	25
1.1.1 Codewars.....	25
1.1.2 CodinGame	27
1.1.3 Points relevés	27
1.2 Gamification	28
1.2.1 Un système de scores personnels.....	29
1.2.2 Un système d'équipe.....	30
1.2.3 Un système d'accomplissements	30
Chapitre 2 : Conception.....	31
2.1 La problématique	31
2.1.1 Les fonctionnalités	31
2.1.2 La persistance des données	31
2.2 Architecture	32
2.2.1 Le Client.....	32
2.2.2 Le Gateway	32
2.2.3 Le service de compilation	33
2.2.4 La base de données	33
2.2.5 Vision à long terme	33

2.3 Choix technologiques	34
2.3.1 Angular	34
2.3.2 Javalin	35
2.3.3 MongoDB	35
Chapitre 3 : Mise en œuvre	37
3.1 Base de données.....	37
3.1.1 Users	37
3.1.2 Programs	38
3.2 Le Gateway.....	40
3.3 Le client	42
3.3.1 Les composants.....	42
3.3.2 Les services.....	43
3.3.3 Les abonnements.....	44
3.3.4 Les programmes.....	44
3.3.5 Les katas.....	46
3.3.6 L'authentification.....	52
3.4 Le service de compilation.....	53
3.4.1 Mise en œuvre.....	53
3.4.2 Containerisation du code.....	56
3.4.3 Les images	57
3.5 Gestion des ressources et privilèges	61
3.5.1 Json Web Token.....	61
3.5.2 Mise en œuvre.....	63
Conclusion.....	69
Annexes	71
Annexe 1 : Manuel de l'utilisateur	73
Sensei.....	73
Créer un programme	74
Gérer votre programme.....	74
Créer un kata	75
Gérer votre kata.....	77
Créer un goal.....	78
Visualiser vos programmes.....	79
Sponsoriser un nouvel utilisateur.....	79
Monji	80
Visualiser les programmes.....	80
Afficher un programme.....	81

Vous abonner à un programme	83
Accéder et gérer vos abonnements	84
Réaliser un kata	84
Rechercher des programmes	85
N'importe qui	86
Rejoindre DojoHepia	86
S'authentifier à DojoHepia	87
Annexe 2 : Ajouter un langage pour les katas.....	89
Introduction	89
Image docker.....	89
Mise à jour du Service de compilation Javalin	91
Mise à jour du Client Angular	92
Tests	93
Publication	93
Annexe 3 : Les services Angular.....	95
Les programmes	96
Les katas	97
La compilation.....	98
L'authentification	98
LANG	98
Annexe 4 : Versionning.....	99
Annexe 5 : Modules utilisés par DojoHepia	101
Annexe 6 : Utiliser DojoHepia.....	103
1. La base de données	103
2. Le Gateway.....	103
3. Le Client	104
4 Le service de compilation.....	104
4.1 Téléchargement de l'image pour le container java	104
4.2 Téléchargement de l'image pour le container python.....	104
4.3 Execution	104
5 Liste des utilisateurs déjà présent	104
Références documentaires.....	105

メメント・モリ

REMERCIEMENTS

Je remercie Delya qui aura rendu ces trois mois plus facile, et qui aura relu ce mémoire avec attention et dévotion.

Je remercie ma mère et mon père qui auront pris le temps de me soutenir, de m'aider, et de lire ce mémoire.

Je remercie mes amis Yann et Gabriela qui m'ont soutenu lors de ce projet, malgré mon indisponibilité.

ÉNONCÉ DU SUJET

HEPIA «Dojo» a pour but de réaliser un outil pédagogique pour l'apprentissage de langages de programmation. Il se base sur des idées existantes, non libres, comme « Codewars » et sur le principe des arts martiaux où les « Katas » permettent de répéter des mouvements connus pour perfectionner sa technique et de l'utiliser de manière instinctive lors d'une situation réelle. L'effort de ce projet est mis sur la qualité du code, la modularité, sa réutilisation et sa maintenance. Le principe de « Gamification » sera utilisé pour fidéliser l'étudiant-e (nombre de points, progression...). Les langages proposés seront : Java11, Python3, SQL avec MySQL et éventuellement un parseur d'algèbre relationnel.

Travail demandé :

Dans les limites du temps disponible :

- Réalisation d'une application client-serveur avec persistance de données
- Authentification facilitée des utilisateurs à l'aide de switch AAI
- Élaboration d'un programme de plusieurs exercices (« Katas ») pour un pédagogue
- Réalisation et exécution de l'exercice par l'étudiant-e avec retours sur
 - les éventuelles erreurs de compilation et warnings
 - les détails éventuels d'un « linter » (convention de nommage, style...)
 - les tests unitaires qui ont échoués

Candidat :

VANINI Alexandre

Filière d'études : ITI

Professeurs responsables :

CAVAT Joël

MALASPINAS Orestis

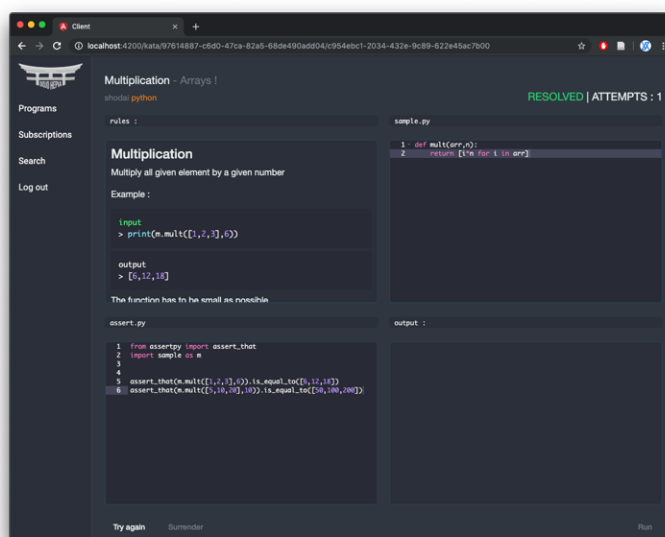
Travail de bachelor soumis à une convention
de stage en entreprise : non

Travail de bachelor soumis à un contrat de
confidentialité : non

RÉSUMÉ

La constante évolution des technologies est de plus en plus flagrante et aujourd'hui, une multitude d'outils voit le jour. Par ailleurs ces outils se sont rendus indispensables et sont maintenant omniprésents dans nos vies et notre société. Cette croissance a permis d'orienter naturellement l'apprentissage scolaire vers une migration informatique de masse, donnant aux élèves la possibilité d'accéder à une multitude d'outils et de techniques leur permettant de graduellement améliorer leur apprentissage de compétences et connaissances. Hepia n'a pas dérogé à cette croissance et se voit dotée de plus en plus de cours utilisant des technologies liées à l'informatique (cours et démonstration sur ordinateur, impression 3D, etc..). Même la filière des technologies de l'information, ITI, met constamment à jour ses outils d'apprentissage pour rester à niveau, et c'est dans cette optique qu'a été imaginé DojoHepia.

DojoHepia est une plateforme d'apprentissage de programmation en ligne, dont l'objectif principal est de permettre à Hepia et, dans le future, d'autres établissements, de réunir ses exercices de programmation sur une plateforme. Le but est de donner la possibilité aux élèves de réaliser des exercices ne nécessitant pas d'installer d'outils ou de Frameworks pour leur réalisation, mais bien d'avoir son propre outil de programmation et de validation en ligne. L'objectif est aussi de leur permettre d'y sauvegarder leur progression générale sur une seule et même plateforme. Ne partant d'aucunes bases et d'aucun projet, une partie du travail se porte sur l'analyse des outils nécessaires et du marché existant. Aussi, une attention toute particulière est portée sur l'étude des technologies de développement Web les plus modernes du moment, ainsi que sur le design et l'expérience utilisateur, pour permettre à DojoHepia de devenir et rester une plateforme ludique et simple d'utilisation.



Candidat :

VANINI ALEXANDRE

Filière d'études : ITI

Professeurs responsables :

CAVAT JOËL
MALASPINAS ORESTIS

Travail de bachelor soumis à une convention de stage en entreprise non

Travail soumis à un contrat de confidentialité : non

TABLE DES ILLUSTRATIONS

Figure 1 - Le Scrum Board (AV, 2019)	23
Figure 2 - Le site codewars (Codewars, 2019).....	26
Figure 3 - Le site CodinGame (CodinGame, 2019).....	27
Figure 4 - Architecture de DojoHepia (AV, 2019)	32
Figure 5 - Exemple d'un système de worker (AV, 2019).....	34
Figure 6 - La collection Users et ses sous-collections (QuickDataBaseDiagrams)	37
Figure 7 - La collection Programs et sa sous-collection (QuickDataBaseDiagrams)	38
Figure 8 - Le client, page d'accueil (AV, 2019).....	42
Figure 9 - Vue des programmes (AV, 2019).....	44
Figure 10 - Carte d'un programme (AV, 2019).....	45
Figure 11 - Vue des programmes, aucun filtre (AV, 2019)	45
Figure 12 - Vue des programmes, Filtre sur tag "list" appliqué (AV, 2019)	46
Figure 13 - Vue des katas, Sensei (AV, 2019)	46
Figure 14 - Une carte kata (AV, 2019).....	47
Figure 15 - Vue de la réalisation d'un kata (AV, 2019)	48
Figure 16 - Interface de création d'un kata (AV, 2019)	51
Figure 17 - Réalisation d'un kata, communication entre les services (AV, 2019).....	60
Figure 18 - Cycle de vie d'une requête HTTP (AV, 2019).....	65
Figure 19 - Manuel utilisateur : N'importe qui - Carte d'un programme (AV, 2019)	73
Figure 20 - Manuel utilisateur : N'importe qui - Carte d'un kata (AV, 2019).....	73
Figure 21 - Manuel utilisateur : Sensei - Page d'un programme (AV, 2019).....	74
Figure 22 - Manuel utilisateur : Sensei - Menu de gestion d'un programme (AV, 2019).....	74
Figure 23 - Manuel utilisateur : Sensei - Menu de duplication d'un programme (AV, 2019) .	75
Figure 24 - Manuel utilisateur : Sensei - Menu de création d'un kata (AV, 2019)	76
Figure 25 - Manuel utilisateur : Sensei - Création d'un kata (AV, 2019)	77
Figure 26 - Manuel utilisateur : Sensei - Menu de gestion d'un kata (AV, 2019).....	77
Figure 27 - Manuel utilisateur : Sensei - Page de Sponsoring (AV, 2019).....	79
Figure 28 - Manuel utilisateur : Sensei - Token généré (AV, 2019).....	79
Figure 29 - Manuel utilisateur : Monji - Tous les programmes disponibles (AV, 2019).....	80
Figure 30 - Manuel utilisateur : Monji - Tous les programmes disponibles Aucun filtre (AV, 2019).....	81
Figure 31 - Manuel utilisateur : Monji -Page : Tous les programmes disponibles Filtrée par tag (AV, 2019).....	81
Figure 32 - Manuel utilisateur : Monji - Page d'un kata, utilisateur non-abonné (AV, 2019) .	82
Figure 33 - Manuel utilisateur : Monji - Page d'un kata, utilisateur abonné (AV, 2019).....	82
Figure 34 - Manuel utilisateur : Monji - Interface de réalisation d'un kata (AV, 2019)	84
Figure 35 - Manuel utilisateur : N'importe qui - Page de création de compte (AV, 2019).....	86
Figure 36 - Manuel utilisateur : N'importe qui - Page d'authentification (AV, 2019)	87

INDEX DES TABLEAUX

Tableau 1 - Vocabulaire de DojoHepia.....	24
Tableau 2 - Les programmes par notion / par langage.....	45
Tableau 3 - Les différents status d'un kata.....	48
Tableau 4 - Le contenu des canevas, création d'un kata.....	52
Tableau 5 - Exemple d'utilisation d'assert py.....	54
Tableau 6 - Résultat de l'exécution d'un kata, assert py.....	54
Tableau 7 - Exemple d'utilisation de Junit 5.....	55
Tableau 8 - Résultat de l'exécution d'un kata, Junit 5.....	55
Tableau 9 - Avantages et inconvénients de la containerisation de l'exécution.	56
Tableau 10 - Mesures, Exécution containerisée vs hostée.....	57
Tableau 11 - Contenu du body, Client -> Gateway -> Service de compilation.....	61
Tableau 12 - Contenu du body, Service de compilation -> Gateway -> Client.....	61
Tableau 13 - Payload d'un JWT.....	63
Tableau 14 - Réponse du Gateway, à l'accès d'une ressource.....	65
Tableau 15 - Liste des états d'un kata.....	83
Tableau 16 - Utiliser DojoHepia - Utilisateurs déjà présents.....	104

TABLE DES ANNEXES

Annexe 1 : Manuel de l'utilisateur.....	73
Annexe 2 : Ajouter un langage pour les katas.....	89
Annexe 3 : Les services Angular.....	95
Annexe 4 : Versionning.....	99
Annexe 5 : Modules utilisés par DojoHepia.....	101
Annexe 6 : Utiliser DojoHepia.....	103

TABLE DES LISTING

Listing 1 - Exemple de listing	24
Listing 2 - Exemple d'utilisation d'un Future (Source : Joël Cavat)	41
Listing 3 - Exemple d'une utilisation de future dans le Gateway	41
Listing 4 - Exemple d'abonnement à une requête, Angular	41
Listing 5 - Dockerfile de l'image python.....	57
Listing 6 - Dockerfile de l'image java	58
Listing 7 - Contenu du fichier .bashrc.....	58
Listing 8 - Contenu du fichier java_test.sh	58
Listing 9 - Commande docker, permettant de créer et lancer un container python	59
Listing 10 - Commande docker, permettant de créer et lancer un container java.....	59
Listing 11 - Un JWT.....	62
Listing 12 - Header d'un JWT.....	62
Listing 13 - Exemple du payload d'un JWT	63
Listing 14 - Dockerfile, python	90
Listing 15 - Exemple d'ajout d'une nouvelle entrée dans le dictionnaire des langages.....	91
Listing 16 - Entrée d'un langage dans le fichier LANG.....	92
Listing 17 - Ajout d'un kata, le code, avant l'ajout d'un langage.....	93
Listing 18 - Ajout d'un kata, le code, après l'ajout d'un langage.....	93
Listing 19 : Utiliser DojoHepia - Commande pour lancer la base de données	103
Listing 20 - Utiliser DojoHepia - Compiler et lancer le Gateway	103
Listing 21 - Utiliser DojoHepia - Compiler et lancer le Client.....	104
Listing 22 - Utiliser DojoHepia - Télécharger l'image pour le container java.....	104
Listing 23 - Utiliser DojoHepia - Télécharger l'image pour le container python	104
Listing 24 - Utiliser DojoHepia - Compiler et lancer le service de compilation	104

LISTE DES ACRONYMES

Acronyme	Signification
AV	Alexandre Vanini

INTRODUCTION

La forte avancée technologique de ces dernières années a poussé tous les domaines à mettre à jour leurs manières de fonctionner, et aucun n'a été épargné. Certains de ces domaines disparaissent progressivement, certains arrivent à tenir le cap malgré la concurrence de l'automatisation, d'autres métiers meurent pour donner naissance à de nouveaux. En revanche, beaucoup d'autres domaines se voient impactés de manière positive, et c'est le cas de l'apprentissage scolaire. En effet, aujourd'hui il existe beaucoup d'enseignements 2.0, adaptés à la technologie web comme par exemple l'apprentissage inversé qui consiste à faire apprendre l'élève chez lui, sur une plateforme en ligne, puis de pratiquer la matière acquise en classe. C'est ce qu'a pour objectif DojoHepia ; mettre à jour les outils de travail des écoles en s'alliant avec les nouvelles technologies. Cette optique d'aider Hepia à faire évoluer ses outils technologiques m'a beaucoup influencé pour le choix de ce sujet. En effet, étant moi-même élève dans cet établissement, je comprends la nécessité qu'aurait une plateforme telle que DojoHepia au sein de son enseignement.

Aujourd'hui, beaucoup d'outils similaires à DojoHepia ont vu le jour sur le Web, tels qu'Openclassroom, CodinGame ou encore Codewars. Ces outils, bien que très différents les uns des autres, sont des mines d'or de connaissances. De plus, ils sont tous des concurrents directs à l'apprentissage scolaire (orienté informatique), qu'on le veuille ou non. C'est aussi pourquoi les écoles, et plus particulièrement Hepia (dans cette thèse), doivent se mettre à jour et se doter d'outils performants pour l'apprentissage afin d'éviter de devenir obsolète face à l'enseignement 2.0.

Pour contrer ces plateformes d'apprentissage en ligne il faut une plateforme robuste, qui s'inspire du présent sans en répéter ses erreurs, et qui a une vision sur le long terme. Et c'est ce que j'ai pour vocation de faire avec DojoHepia. Depuis que je fais de l'informatique, je ne cesse de m'intéresser à comprendre pourquoi certains sites internet sont faits de telle ou telle manière, et pourquoi les informations s'affichent de cette façon. Mais aussi quelles sont les meilleurs moyens de simplifier au maximum la vie de l'utilisateur et de rendre son voyage plus intéressant.

L'objectif principal de ce mémoire est dans un premier temps de comprendre comment les autres plateformes fonctionnent, et comment elles approchent le côté pédagogique de leur partage de connaissances. L'objectif est aussi de décrire et de détailler l'ensemble du développement de DojoHepia, de son design, sa conception et sa mise en œuvre.

Le premier chapitre est destiné à l'étude des plateformes d'apprentissage en ligne, CodinGame et Codewars. J'expliquerai les résultats de mon étude et les éléments que j'ai relevés, des avantages et inconvénients à l'expérience utilisateur. De plus, ce chapitre introduira le système de Gamification que DojoHepia devra adopter pour fidéliser de manière optimale ses utilisateurs et rendre l'expérience encore plus intéressante.

Le second chapitre a pour but de décrire les fonctionnalités et la conception de DojoHepia. Je parlerai des décisions architecturales et de leur développement sur le long terme. Mais aussi des choix technologiques faits pour la plateforme.

Le troisième et dernier grand chapitre porte son attention sur la mise en œuvre de la plateforme, et de tous ses éléments techniques. J'aborderai avec plus de détails la façon dont a été mis en place l'implémentation de DojoHepia, comment elle réussit à remplir son objectif de plateforme d'apprentissage en ligne, et comment fonctionnent les interactions entre les divers services et composants. J'aborderai aussi les choix de conception graphique et d'agencement de l'information qui ont été pris pour que ma plateforme se voie dotée de la meilleure expérience utilisateur possible.

DOJOHEPIA

Dans les arts martiaux japonais, un kata est une série de mouvements que répète régulièrement un élève pour l'apprendre à la perfection. En informatique, le kata est un exercice qu'un programmeur se doit de répéter souvent pour ne pas perdre les bases d'un langage. Par exemple, il doit chaque mois créer un système de piles avec pointeur en C, pour que cela devienne un automatisme. De cette manière, le programmeur ne perd pas de temps sur ces problèmes lorsqu'un nouveau projet lui est attribué. C'est grâce à cette notion de « kata » que ma plateforme a porté son vocabulaire sur les arts martiaux japonais et notamment Le Shōtōkan-ryū (c'est aussi pourquoi DojoHepia contient le mot « dojo », lieu où l'on pratique les arts martiaux).

Lexique des arts martiaux :

sainteskarateclub.com/non-categorise/lexique-des-arts-martiaux.html

MÉTHODOLOGIE DE TRAVAIL

Dans le cadre du projet de bachelor et dans l'objectif de nous rapprocher d'un projet d'entreprise moderne, Mr Cavat nous a amené à adopter un mode de travail bien spécifique, que l'on pourrait apparenter à la méthode agile "Scrum" (scrum.org, (*What is Scrum?* 2019)).

User Story Mapping

Le développement était séparé en itérations ce qui a permis de choisir les fonctionnalités que j'allais implémenter à chaque nouvelle version. De plus, pour permettre de correctement structurer ces itérations, et retenir l'état du développement au fil des semaines, Mr Cavat et moi, avons mis en place un "User Story Mapping" (voir Figure 1), qui est un tableau physique, rempli de tâches sous forme de post-it, triés et catégorisés. Il y avait 4 catégories :

- 1) Must Have
- 2) Should Have
- 3) Nice To Have
- 4) Won't Have

qui permettaient de maximiser la valeur ajoutée à chaque nouvelles itérations.

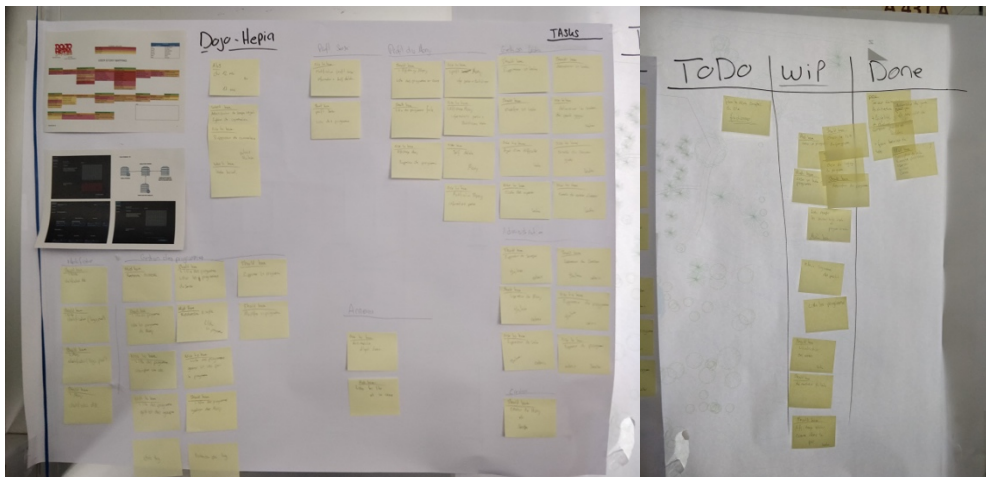


Figure 1 - Le Scrum Board (AV, 2019)

Les différentes catégories (à droite, sur la Figure 1) permettaient de suivre le fil du développement de l'itération en cours, de valider les tâches terminées et de décider quelles seraient les nouvelles tâches pour la prochaine itération.

AVANT-PROPOS

Ce court sous-chapitre est destiné à expliquer certains détails importants dans le contenu de cette thèse.

Vocabulaire

Comme mentionné dans l'introduction, DojoHepia à un vocabulaire particulier. En effet, la plateforme voulant se rapprocher le plus de l'esprit « dojo » (lieu de respect et d'écoute), Mr Cavat et moi-même avons décidé d'y intégrer un vocabulaire se rapprochant du Shōtōkan-ryū. Vous serez donc amené à croiser des mots comme « Sensei » ou « Monji ». Le Tableau 1 a été écrit dans le but de traduire le vocabulaire dans un langage plus professionnel.

Vocabulaire	
Shōtōkan-ryū.	Dojo Hepia
Dojo	Le site internet
Shodai	Fondateur
Sensei	Professeur
Monji	Elève
Kata	Exercice

Tableau 1 - Vocabulaire de DojoHepia

Figures

Pour simplifier la lecture de cette thèse, les chapitres sont généralement accompagnés d'images de la plateforme DojoHepia. Ces images ne sont pas représentatives de ce qu'est la plateforme dans son entièreté car elles ont subi des modifications et des rognages.

Listing

Certains paragraphes décrivant le fonctionnement de la plateforme sous souvent accompagnés d'encadrés de code. Ces bouts de codes sont en anglais et proviennent du code source de DojoHepia. Ils sont stylisés de deux manières :

<pre>export const EXAMPLE: Style _1[] = [{ id: 'some code', }, {...}];</pre>	<p>Code style 2, exemple</p> <p># Ligne 1</p> <p># Ligne 2</p>
--	--

Listing 1 - Exemple de listing

Et seront toujours légendés « Listing *N* »

CHAPITRE 1 : ANALYSE FONCTIONNELLE

1.1 ANALYSE DU MARCHÉ ET DES BESOINS

Pour que DojoHepia devienne une bonne plateforme, je suis allé voir ce qui se faisait déjà sur le marché. Non seulement pour m'en inspirer (une bonne idée reste une bonne idée), mais surtout pour ne pas reproduire les erreurs des autres plateformes. Le marché actuel est scindé en plusieurs grandes plateformes telle que Codewars ou encore CodinGame. Il existe d'autres plateformes qui sont plus orientées sur l'apprentissage théorique (comme Openclassroom) plutôt que l'apprentissage pratique mais je ne les aborderai pas car, rappelons-le, l'objectif de DojoHepia est d'être une plateforme d'apprentissage de programmation où les utilisateurs ont la possibilité de **réaliser** les exercices proposés.

Les points importants que je vais relever lors de cette analyse sont les suivants :

- L'expérience utilisateur
- La fidélisation des utilisateurs

J'aborde ces points à travers l'analyse des sites Codewars (codewars.com) et CodinGame (codinggame.com), qui sont les sites internet qui se rapprochent le plus de ce que DojoHepia a pour volonté d'être.

1.1.1 Codewars

Codewars a déjà la notion de dojo et de kata, que je cherche à faire apparaître sur ma plateforme. De plus, cette plateforme est axée sur un système de compétition où les utilisateurs se confrontent sur des katas. Ces mêmes katas leurs permettent de gagner en kyu, qui est un grade utilisé dans les arts martiaux. Plus un utilisateur effectue de katas, plus il monte en grade. Dans le Judo traditionnel et ses dérivés, ce système permet de catégoriser les élèves. Il est surtout présent dans le Judo européen (au Japon, pays originaire du Judo traditionnel, il existe soit des maîtres, soit des apprenants). Dans le Judo moderne, les Kyus et les Dans (à prononcer « *dæn* ») sont les moyens officiels d'imposer une hiérarchie, mais les élèves sont plus habitués à recevoir des ceintures de différentes couleurs. Lors de la réalisation d'un kata avec succès, l'utilisateur gagne des points qui lui permettent d'évoluer dans un tableau des scores, et c'est là que se trouve tout l'aspect compétitif de Codewars.

L'interface est très chargée et il est très compliqué de comprendre ou de s'orienter lorsqu'on arrive la première fois sur le site. Les informations importantes ne sont pas mises en avant et cela a pour conséquences de bloquer l'utilisateur sur la page un certains nombres de secondes avant qu'il ne trouve ce qu'il cherche. Pour résumer, le site n'est pas "beginner-friendly" (compliqué à prendre en main pour un débutant). En revanche, le système de clans est très intéressant ; il pousse les utilisateurs à créer des équipes et à faire de leur mieux pour battre les autres clans. Le fait que Codewars implémente un système de niveaux avec rappel aux arts-martiaux (les Kyus et les Dans) est très important car il permet de catégoriser les utilisateurs d'une manière à ce que les comparaisons ne deviennent pas péjoratives. En effet, être d'un grade inférieur signifie juste que le niveau de l'utilisateur n'est pas adapté à certains katas, et non pas qu'il est moins bon qu'un autre utilisateur (très représentatif de la mentalité du Judo par exemple). Cette distinction est minime, mais pourtant très importante. De plus, cette plateforme comporte un système très intéressant de commentaires. En effet, à la manière de GitHub, les utilisateurs sont capables d'ouvrir des "Issues" (à traduire par "Problèmes") qui leurs permettent de mettre en avant un problème sur un kata, ou tout simplement de donner un avis sur celui-ci. Une dernière chose très intéressante, est que lorsque l'on abandonne ou que l'on réussit un exercice, l'utilisateur a accès à un classement des meilleures solutions proposées par les autres utilisateurs, ceci permet de juger sa solution et de comprendre ce qu'on peut y améliorer.

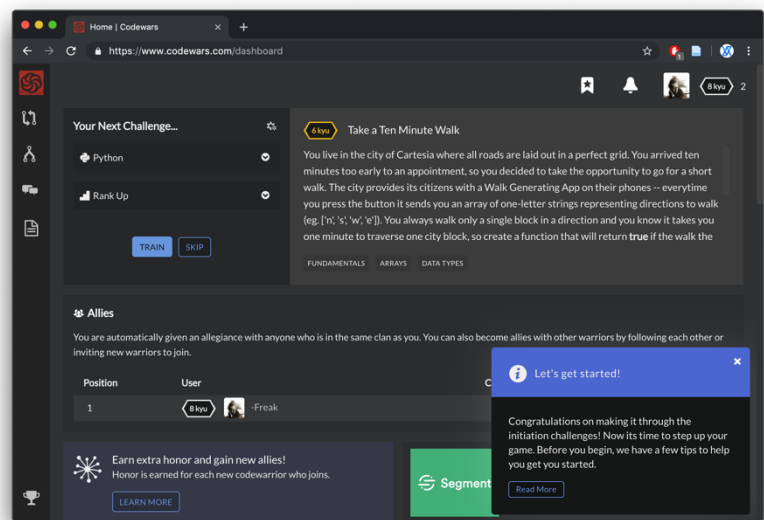


Figure 2 - Le site codewars (Codewars, 2019)

1.1.2 CodinGame

CodinGame est un site d'apprentissage en ligne axé jeux vidéo. Le but est de résoudre des puzzles ou des niveaux de jeux en écrivant soi-même les lignes du code. Malgré un système d'aide lors de l'arrivée d'un utilisateur, l'interface, tout comme celle de Codewars, est très dispersée et incompréhensible au premier abord. En revanche, l'aspect ludique de la plateforme renforce vraiment la fidélisation des utilisateurs. En effet, ceux-ci sont poussés à voir la

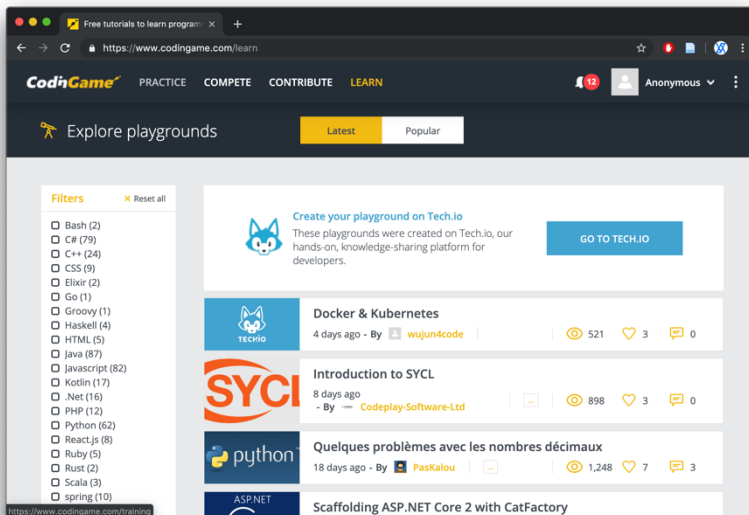


Figure 3 - Le site CodinGame (CodinGame, 2019)

qui offre à l'utilisateur un moyen de personnaliser son espace d'entraînement (même si ce n'est que très minime).

Comme CodinGame est axé sur l'esprit jeu vidéo ludique, l'utilisateur a la possibilité de réaliser des succès (tel que "Premier pas !" par exemple), qui permet de donner des objectifs secondaires à réaliser pour les utilisateurs. De plus, la plateforme se voit dotée d'un système d'amis, qui permet de comparer son avancement et de visualiser leurs succès (Cela génère un petit système de compétition).

1.1.3 Points relevés

Comme l'analyse de Codewars et CodinGame l'a relevé, l'expérience utilisateur est un point très important pour que DojoHepia soit une bonne plateforme. En effet, trouver le juste milieu entre une excellente expérience utilisateur et un site complet et fonctionnel est primordial. Certains points importants devront tôt ou tard être implémentés pour DojoHepia afin de rendre la plateforme encore plus fonctionnelle et plus agréable à utiliser. Notamment un système

plateforme comme un lieu où passer du temps et s'amuser plutôt qu'une plateforme pour s'exercer à programmer, même si ça n'en reste pas moins l'objectif du site.

Une fonctionnalité non primordiale mais qui reste pas intéressante, est que l'utilisateur peut changer le thème du site entre Light et Dark (lumineux et sombre), ce

général de clans et de tableaux des scores où les utilisateurs de l'application pourront s'affronter pour monter dans le classement, en solo ou en équipe. Il faudra toutefois modérer le côté compétitif pour éviter que cela ne devienne ingérable et que les joueurs se mettent à tricher pour devenir meilleurs que les autres équipes, car l'objectif de notre plateforme est l'apprentissage avant tout. Monter en grade implique un système de points gagnés et d'obtention de grade, qui permettent de se rapprocher d'un outil ludique où l'apprenant apprécie passer du temps. Ce système permettrait de débloquer la fonctionnalité de création de katas aux utilisateurs ayant le grade minimum requis.

Un dernier point qui permettrait de rendre la plateforme plus vivante est l'implémentation d'un système de commentaires et de liste d'amis. Ces deux fonctionnalités permettraient aux utilisateurs de noter les exercices auxquels ils participent, mettre en avant des points importants, mais aussi de conseiller les nouveaux apprenants. Quant à la liste d'amis, un système complet permettrait d'accéder à l'avancement de nos amis, et de pouvoir leurs parler en temps réel à l'aide d'une messagerie.

1.2 GAMIFICATION

La « Gamification » (ou ludification, en français), est l'utilisation de mécanismes de jeux et de récompenses permettant de rendre ludique des activités de tous genres. Très appliquée, et surtout en informatique, elle aide à fidéliser l'utilisateur à la plateforme en lui donnant envie d'y revenir pour s'améliorer.

L'état actuel de la plateforme n'inclut aucun de ces mécanismes de gamification pour la simple et bonne raison que l'ensemble des tâches concernant cette partie de DojoHepia est resté dans les non-faits, au détriment de tâches plus importantes.

Toutefois, à l'aide de l'analyse du marché et des points relevés, j'ai pu dresser une liste de fonctionnalités que DojoHepia pourrait développer afin de rendre la plateforme plus intéressante et ludique :

- Un système de score personnel

- Un tableau des scores
- Un système de rangs
- Fonctionnalités à débloquent
- Un système d'équipes
- Un système d'accomplissement

1.2.1 Un système de scores personnels

Un système de scores permet de pouvoir donner des points à un utilisateur. De ce fait, la difficulté d'un kata déterminerait le nombre de points à attribuer. Psychologiquement un utilisateur dont le nombre de points augmente de manière conséquente quand il résout un kata avec une haute difficulté, se sentira récompensé de ses efforts.

Un tableau des scores

Ce système de points classierait les utilisateurs dans un tableau, leur donnant ainsi la possibilité d'afficher « leur supériorité » aux autres (toujours dans un esprit de respect compétitif). L'effet serait motivant et inciterait ceux aux scores moins élevés, à effectuer plus de katas à difficulté accrue pour ainsi grimper dans le classement.

Dans une plateforme d'apprentissage en ligne comme DojoHepia, un système pareil ne devrait pas s'orienter sur tous les katas de manière générale car les utilisateurs ne sont invités qu'à résoudre les katas de leur Sensei. Il faudrait donc avoir un système de classement par programme, permettant ainsi de catégoriser les utilisateur sur une notion. Il faudrait faire attention toutefois à ne pas pousser le concept trop loin car, comme relevé dans l'analyse du marché, une forte compétition pourrait inviter les utilisateurs à tricher et à s'éloigner de l'objectif principal, c'est à dire, l'apprentissage.

Un système de rang

Le système de points donnerait lieu à un système de rang général (par exemple débutant, intermédiaire, avancé) basé sur la réussite des katas par utilisateur. Ce rang indiquerait l'avancée d'un utilisateur ainsi que le niveau de ses compétences. Ce système pourrait s'avérer

particulièrement pertinent avec la fonctionnalité de commentaires vu qu'il permettrait de créditer et comparer les avis des différents utilisateurs.

Fonctionnalités à débloquent

L'utilité d'un système de scores personnels ne s'arrête pas qu'à montrer aux autres que l'on a été meilleur ou moins bon sur un programme. En effet, on pourrait imaginer un système de fonctionnalité à débloquent en fonction de son nombre de points ou de son rang. Par exemple, un utilisateur fidèle et compétant, ayant atteint un rang élevé pourrait se voir offrir le rang de « Sensei », lui permettant par la suite de créer ses propres katas.

1.2.2 Un système d'équipe

Un système d'équipe pourrait aussi être mis en place afin que les utilisateurs puissent résoudre des katas en équipe. Cela leur permettrait de se confronter en groupe à d'autres personnes, renforçant par la même occasion l'esprit de cohésion au sein d'une classe. A la manière du système de scores personnels, les équipes seraient affichées dans un tableau de scores, permettant d'augmenter l'esprit de compétition et de rendre la réalisation des katas encore plus intéressante.

1.2.3 Un système d'accomplissements

Très présent dans les jeux vidéo aujourd'hui, les systèmes d'accomplissements (plus connus sous leur appellation anglaise « Achievements », ou encore « Trophées »), permettent de donner des objectifs à l'utilisateur. Ces objectifs à réaliser feraient gagner des points, et donc, gagner en rang. Ces accomplissements seraient visibles par les autres utilisateurs et donnerait l'opportunité de valoriser son compte avec l'acquisition de trophées rares.

CHAPITRE 2 : CONCEPTION

2.1 LA PROBLÉMATIQUE

DojoHepia n'est basé sur aucun autre projet. Ses seules prémisses sont les analyses de marché énoncées plus haut.

2.1.1 Les fonctionnalités

Le but de DojoHepia est de proposer une réalisation d'exercices en ligne. Pour ce faire, j'ai imaginé un système de *programme*. Un *programme* est un ensemble de katas sur un langage en particulier. Quant-aux katas, ils sont des exercices de programmation proposés par un Sensei, qu'un utilisateur réalise à l'aide d'un *outil de programmation* en ligne. De plus, l'utilisateur pourra « suivre » un programme à l'aide d'un système d'abonnement, et pourra ainsi y sauvegarder sa progression.

DojoHepia aura donc 5 composantes principales :

- Un système d'abonnement
- Les programmes
- Les katas
- Un système d'authentification
- Un outil de programmation

Ces composantes devront proposer des sous-ensembles de fonctionnalités pour que les utilisateurs de la plateforme puissent profiter au maximum de l'expérience souhaitée. Pour la question de l'authentification, il faudra imaginer un système de passes où seuls ceux en leur possession pourront s'identifier sur DojoHepia afin qu'elle reste une plateforme composée exclusivement d'utilisateurs académiques.

2.1.2 La persistance des données

Il faut aussi que DojoHepia se dote d'un système de persistance de données, pour ainsi stocker les programmes, les katas, ou encore la progression d'un utilisateur.

2.2 ARCHITECTURE

Pour pallier à la problématique énoncée un sous chapitre plus haut, l'architecture de DojoHepia se voit constituée de 4 composants (voir Figure 4)

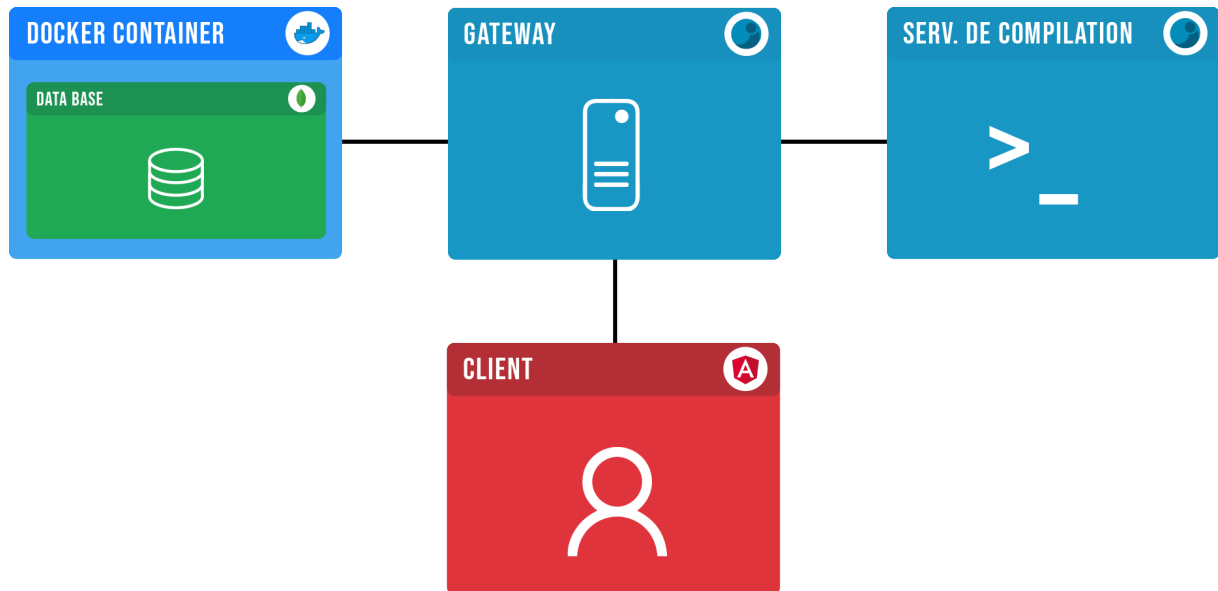


Figure 4 - Architecture de DojoHepia (AV, 2019)

2.2.1 Le Client

Le Client est le front-end de DojoHepia. Il est implémenté avec Angular CLI, ce qui permet de le rendre plus robuste étant donné que c'est un langage compilé. Ce service permet à l'utilisateur d'interagir avec la plateforme.

2.2.2 Le Gateway

Le Gateway est un serveur HTTP, en Java, implémenté à l'aide du Framework Javalin. Ce service peut-être décrit comme l'API de DojoHepia, faisant interface entre tous les autres services. De plus, il est basé sur l'architecture REST pour être dans les standards et faire preuve de robustesse. Aussi, ce service est dit « stateless », c'est à dire que la réponse d'une requête ne dépend pas du moment où elle a été envoyée.

2.2.3 Le service de compilation

Le service de compilation permet de lancer l'exécution du code dans un container pour en rendre le résultat.

2.2.4 La base de données

La base de données fonctionne à l'aide de MongoDB et est containerisée à l'aide de Docker. Elle permet de garder de manière persistante les données de DojoHepia. Elle doit être en mesure de stocker des programmes, des katas ainsi que des utilisateurs et abonnements.

2.2.5 Vision à long terme

Le Gateway et le service de compilation ont été conçus pour être scalables sur le long terme.

« En informatique matérielle et logicielle et en télécommunications, la scalability ou scalabilité (calque de traduction) désigne la capacité d'un produit à s'adapter à un changement d'ordre de grandeur de la demande (montée en charge), en particulier sa capacité à maintenir ses fonctionnalités et ses performances en cas de forte demande. » (Scalability 2018).

Dans l'objectif d'améliorer la plateforme, j'ai décidé de séparer les services sur des serveurs différents. Une telle séparation des services permet de les remplacer facilement. Une des volontés de la plateforme est, dans le futur, d'implémenter un système de « Worker ».

Le système de « Worker »

Ce système se situerait entre le Gateway et le service de compilation. Dans cette implémentation, le Gateway enverrait ses tâches d'exécution dans une file d'attente pour qu'il puisse s'occuper d'autres tâches en attendant de recevoir la réponse du service de compilation. Le service de compilation (que l'on pourrait multiplier) quant à lui, s'occuperait des tâches d'exécution et en restituerait le résultat dans un pipeline de réponses.

La Figure 5 est un schéma d'exemple du système de Worker, adapté à l'architecture de DojoHepia.

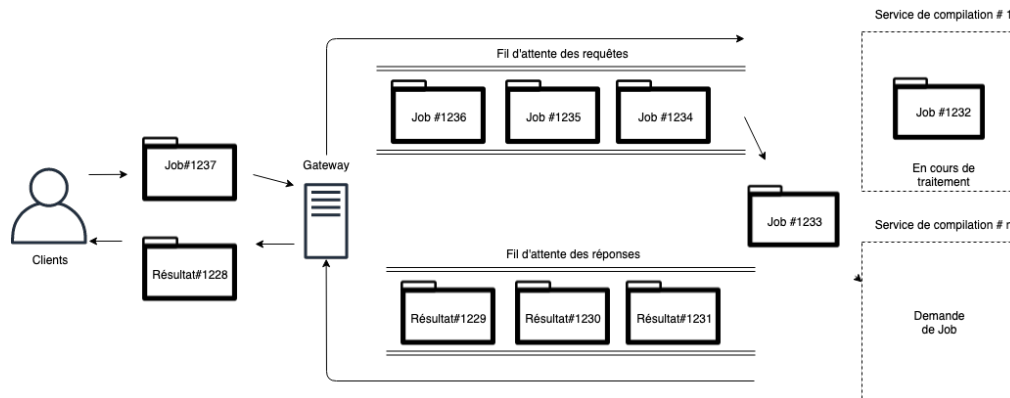


Figure 5 - Exemple d'un système de worker (AV, 2019)

De plus, séparer les services de cette manière permettrait de passer d'une implémentation Java Server à une implémentation Scala Server, ou encore à migrer sur une nouvelle base de données.

2.3 CHOIX TECHNOLOGIQUES

Mes motivations pour ce projet se trouvent aussi dans les choix technologiques. J'avais pour volonté de sortir de ma zone de confort et de me confronter à l'apprentissage de nouvelles techniques. Cela m'a permis aussi d'approfondir mes connaissances sur le développement Web et de me tenir à jour. Pour sortir de cette zone de confort, j'ai dû me séparer de mes Framework favoris (NodeJS, RethinkDB, etc..) pour en explorer de nouveaux. De plus, pour pouvoir créer une application robuste, les conventionnels NodeJS et autres Framework se basant sur du Javascript n'étaient pas de mise, car Javascript est un langage non typé (les variables n'ont pas de type).

2.3.1 Angular

Pour ce qui est de la grande tendance web du moment, c'est Angular (angular.io, (Angular 2019)) qui est sur le devant de la scène. Ce Framework simple et à la fois complet permet de développer un front-end robuste et de confiance. En plus d'être doté d'une bonne documentation et d'un tutoriel complet et compréhensible, ce Framework a été conçu pour la réutilisation et la modularisation, notamment grâce aux "components" et "services". C'est aussi

pour moi le moyen d'aborder un nouveau Framework. La simplicité de mise en place combinée avec un puissant outil créé par Google, font d'Angular un outil de confiance et de qualité.

2.3.2 Javalin

Javalin (javalin.io) est une technologie de serveur basé sur le langage Java. C'est contre cela que j'ai troqué NodeJS, car Java est un puissant langage typé et orienté objet. Dans l'ensemble, Javalin est très similaire à NodeJS, ce qui en fait un outil simple à prendre en main.

2.3.3 MongoDB

Parmi les nombreux choix de bases de données, c'est MongoDB (mongodb.com) qui a retenu mon attention. C'est pour moi une nouvelle technologie qui se rapproche des bases de données non relationnelles que j'ai utilisé dans le passé (notamment RethinkDB). Sur le marché de travail, MongoDB est une qualité très recherchée et pouvoir ajouter cette corde à mon arc en fait un parfait allié. En plus d'être un langage recherché et beaucoup utilisé, MongoDB est puissant et bien documenté (de plus, le langage utilisé dans le MongoSHELL est du Javascript, ce qui son utilisation d'autant plus facile).

CHAPITRE 3 : MISE EN ŒUVRE

3.1 BASE DE DONNÉES

Avant d'entrer dans les détails d'implémentation, il est important de se faire une idée de la base de données, et de comment sont agencés les informations dans celle-ci. Ma base de données est constituée de deux collections (vocabulaire de MongoDB, docs.mongodb.com/manual/reference/sql-comparison/) .

- Users
- Programs

Ces deux collections permettent à elles seules de maintenir toutes les informations du site, car elles contiennent des sous-ensembles de collections.

3.1.1 Users

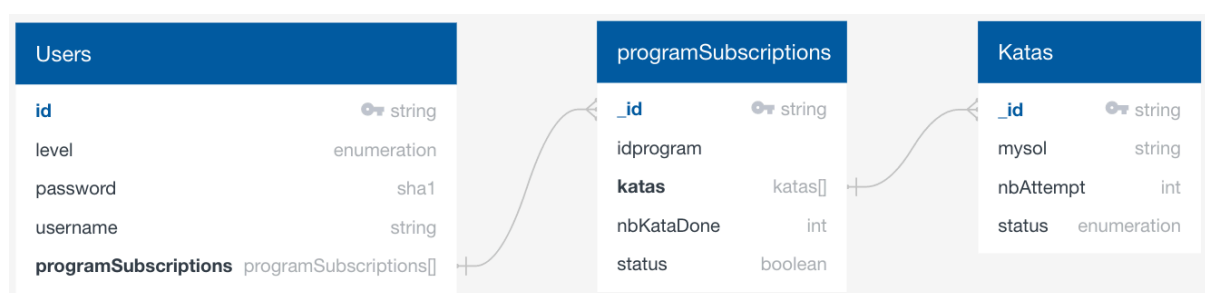


Figure 6 - La collection Users et ses sous-collections (QuickDataBaseDiagrams)

Attention : Pour une meilleure visibilité, les sous collections ont été transformées en table, mais ce sont bien des sous collections de la collection Users.

Voici la collection complète **Users** (Figure 6), composée d'une autre sous collection **programSubscriptions** qui elle-même est composée d'une sous collection **Katas**. Hormis le champ **programSubscriptions** que j'évoque dans le sous-chapitre suivant "*Sous-collections - programsSubscription*", les autres champs de la collection **Users** sont relativement simples¹, et permettent de stocker un utilisateur avec le minimum d'informations nécessaires

¹ Le champ « level » est évoqué dans le chapitre 3 dans la partie « Gestion des ressources et privilèges »

Sous-collection - programsSubscription

`programSubscription` permet à l'utilisateur de s'abonner à un programme et d'y sauvegarder sa progression. Le champ `nbKataDone` permet d'indiquer le nombre de katas qu'un utilisateur a effectués. `Status` quant à lui, permet d'indiquer si un utilisateur est abonné au programme ou non, à noter qu'un utilisateur peut ne *plus* être abonné mais avoir une progression sauvegardée quand même.

Sous-Sous-collection - Katas

La sous-collection de `programSubscription`; `Katas`, permet de stocker la progression de l'utilisateur sur le kata concerné. Du moment que l'utilisateur arrive sur la page de réalisation d'un Kata, un document est créé et le statut passe de "TODO" à "ONGOING". La solution de l'utilisateur une fois le kata terminé, (qu'il soit abandonné ou réussi) sera stockée dans le champ "mysol".

3.1.2 Programs

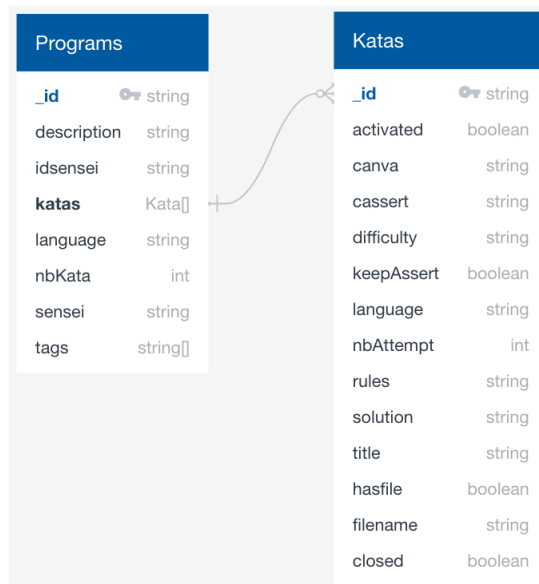


Figure 7 - La collection Programs et sa sous-collection (QuickDataBaseDiagrams)

Attention : Pour une meilleure visibilité, la sous collection a été transformée en table, mais c'est bien une sous collection de la collection Programs

Voici la collection complète “Programs” (Figure 7). Celle-ci permet de stocker toutes les informations nécessaires à la description d’un programme. Le champ *Katas* est une sous-collection que je détaille dans le chapitre « mise en œuvre – Le Client – Les katas » Quant au champ *tags*, il permet de stocker sous la forme d’un tableau les tags qui catégorisent un programme, cela est notamment utilisé pour filtrer les programmes.

Sous-collection - Kata

La sous-collection Kata permet de stocker les informations nécessaires pour décrire ceux-ci. Les champs *canva*, *cassert* et *solution* sont le stricte minimum pour la réalisation d’un kata. Ils permettent respectivement de stocker la base de codes qui sera donnée à l’utilisateur lors de sa tentative de résolution du kata, de stocker la batterie test du kata et de stocker la solution du kata dans le cas où l’utilisateur abandonnerait. Une petite particularité est celle du champ *keepAssert*, il permet de spécifier si le créateur du kata souhaite que la boîte de batterie de tests soit visible pour l’utilisateur lors de la réalisation d’un kata.

3.2 LE GATEWAY

Comme présenté dans l'architecture de DojoHepia, le Gateway est l'élément central de l'application. Il permet de faire la liaison entre chaque service et d'orienter les informations en conséquence. Si ce service porte ce nom, c'est parce que dans une vision à long terme, DojoHepia aimerait se munir d'un vrai Gateway, ne servant qu'à relayer les informations reçues vers les services concernés. En effet, pour le moment, le Gateway contient de la logique applicative, il n'est donc pas réellement un « Gateway » (car son rôle ne se limite pas à diriger l'information).

Mon Gateway est basé sur une architecture REST,

“Pour donner une définition simpliste, l'architecture REST est une interface entre systèmes utilisant le protocole http, utilisée pour obtenir des données et les manipuler (comme du XML ou JSON)” (REST API 2019) – Traduit de l'anglais par Alexandre Vanini.

ce qui induit que la communication entre mes différents services se fait à travers le protocole HTTP (comme décrit dans le Chapitre 1 : Conception). Le protocole HTTP est un protocole de communication synchrone, ce qui signifie que le serveur ne lâchera pas la main sur une requête avant de l'avoir réalisé, bloquant ainsi les autres requêtes entrantes.

Même si Javalin permet déjà de rendre une partie du fonctionnement asynchrone (limité à 200 requêtes à la fois), il est préférable de rendre le flux entre le Client et le Gateway totalement asynchrone, pour s'affranchir de problèmes d'attente en cas de grosse montée en charge. Pour ce faire, j'utilise le mécanisme des Futures en Java. Les Futures sont un des moyens de rendre l'exécution d'un code Java concurrent, ce qui signifie que les requêtes peuvent être exécutées de manière simultanées. Le listing 2 est un exemple d'utilisation des Futures.

```
String test() {
    try{
        Thread.sleep(1000);
    }catch (Exception e){}
    return "Test";
}

// Code qui dure 4 secondes
for (int i = 0; i < 3; i++){
    System.out.println( test() );
}

// Code qui dure 1 seconde
```



```
for (int i = 0; i < 3; i++){
CompletableFuture.supplyAsync( () -> test() ).thenAccept( System.out::println );
//                                     supplier, lazy call           side effect
}
```

Listing 2 - Exemple d'utilisation d'un Future (Source : Joël Cavat)

Javalin implémente nativement un moyen de donner un Future en réponse à une requête (voir Listing 3).

```
// ctx.json(future)

app.get("program/:id", ctx -> {
ctx.json(dbPrograms.programDetailsById(ctx.pathParam("id")).toCompletableFuture());
}, roles(Roles.SHODAI, Roles.SENSEI, Roles.MONJI));
```

Listing 3 - Exemple d'une utilisation de future dans le Gateway

Cela permet d'encapsuler la requête du Client dans un Future, et de le donner à un processus. De cette façon, le client peut continuer de recevoir des requêtes pendant que le processus exécute le code encapsulé. Lorsque le code est exécuté, le résultat est envoyé au client et le processus est de nouveau disponible.

Du côté Client

Angular propose un système d'abonnement à une requête (voir : angular.io/tutorial/toh-pt4), ce qui assure une demande de clients correctement traitée (ce système permet également de gérer d'éventuelles erreurs). Le Listing 4 est un exemple d'abonnement à une requête.

```
// Requête au serveur : Rapatriement d'un programme particulier
this.programService.getById(this.programid).subscribe((data: Program) => {
// Traitement du résultat, lorsque le client reçoit la réponse du Gateway
}, (erreur => {
// Traitement des éventuelles erreurs
}));
```

Listing 4 - Exemple d'abonnement à une requête, Angular

3.3 LE CLIENT

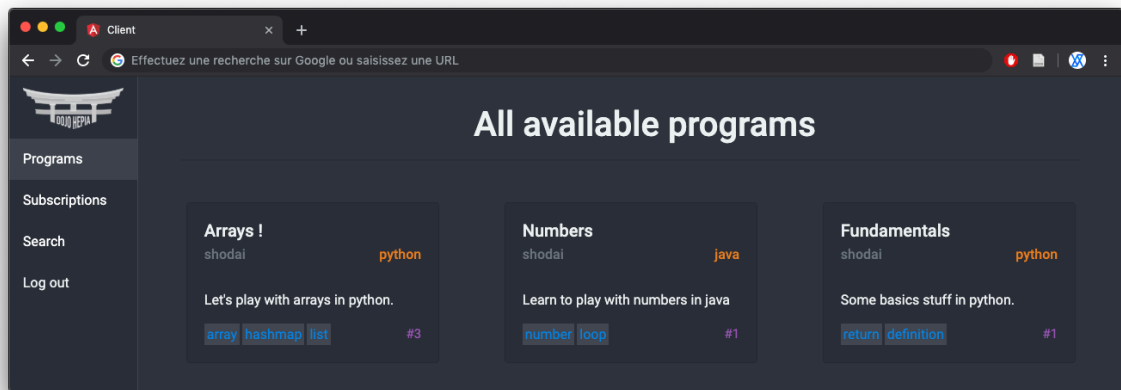


Figure 8 - Le client, page d'accueil (AV, 2019)

Comme décrit dans le sous chapitre « choix technologiques », le Client (l'interface en Figure 8) est implémenté à l'aide du Framework Angular CLI. Ce Framework permet de facilement créer des briques de la plateforme à l'aide des composants, et de créer une interface de communication avec le Gateway à l'aide des services. Ce sous chapitre a pour but, à travers les points suivants, de décrire comment les différentes briques fonctionnent entre elles et quels ont été les choix d'implémentation pour que soit posé l'interface de ma plateforme d'apprentissage de programmation en ligne :

1. Les composants
2. Les services
3. Les abonnements
4. Les programmes
5. Les katas
6. L'authentification

3.3.1 Les composants

Les composants, ou « briques » sont des répertoires contenant, de manière générale, 4 fichiers :

1. Un fichier HTML
2. Un fichier CSS
3. Un fichier typescript

4. Un fichier de spécification (*propre à Angular*).

La particularité d'un composant est qu'il se concentre sur une fonctionnalité. En effet, j'ai par exemple créé un composant permettant d'afficher sous forme de carte (voir Figure 8), la liste des programmes disponibles sur la plateforme. Ce composant, une fois défini (grâce au 4 fichiers cités plus haut), est réutilisable dans n'importe quel autre composant. Cela s'avère très utile car nous évite de réécrire une fonctionnalité déjà existante.

Pour une plus ample description de ce qu'est un composant, rendez-vous sur : angular.io/guide/architecture-components (*Angular - Introduction to components* 2019)

3.3.2 Les services

Les services Angular sont des classes pouvant créer une interface de communication (API) avec le Gateway. Ils donnent la possibilité de contacter celui-ci pour recevoir / poster / mettre à jour ou encore supprimer des informations. On peut catégoriser les services créés pour DojoHepia en 5 parties :

- Les programmes
 - Interactions avec un programme
 - Les abonnements aux programmes
 - La recherche de programmes
- Les katas
 - Interaction avec un kata
 - Les abonnements aux katas
- La compilation
- L'authentification
 - Connection, inscription, génération de token
- Le service LANG
 - Récupération du code canevas lors de la création d'un kata

Note : L'annexe 3 liste et décrit l'ensemble des services, ainsi que leurs fonctions.

3.3.3 Les abonnements

Les abonnements sont des entités invisibles de DojoHepia ; c'est le moyen de sauvegarder la progression d'un utilisateur sur un programme. Ils permettent notamment de :

- Garder un suivi de son évolution sur un programme spécifique (nombre de katas réalisés)
- Garder un suivi de son évolution sur un kata (nombre d'essai, solution, etc..)
- Protéger un programme à l'aide d'un mot de passe
- Avoir une page "Mes abonnements"

Sur le long terme

Sur un plus long terme, un système d'abonnement donnerait la possibilité de récupérer les statistiques d'un programme tout entier, ou d'un kata particulier. On pourrait par exemple imaginer que le Sensei souhaite voir les solutions de ses abonnés sur un kata, ainsi que le taux de réussite. Il aurait donc l'occasion de jauger la qualité de son enseignement et de savoir quand adresser un élève (en cas de résultats peu convaincants, par exemple).

3.3.4 Les programmes

Les programmes sont les points centraux de DojoHepia. En effet, ce sont ces entités qui relient les différentes parties entre elles, que ce soit les katas ou le système d'utilisateurs.

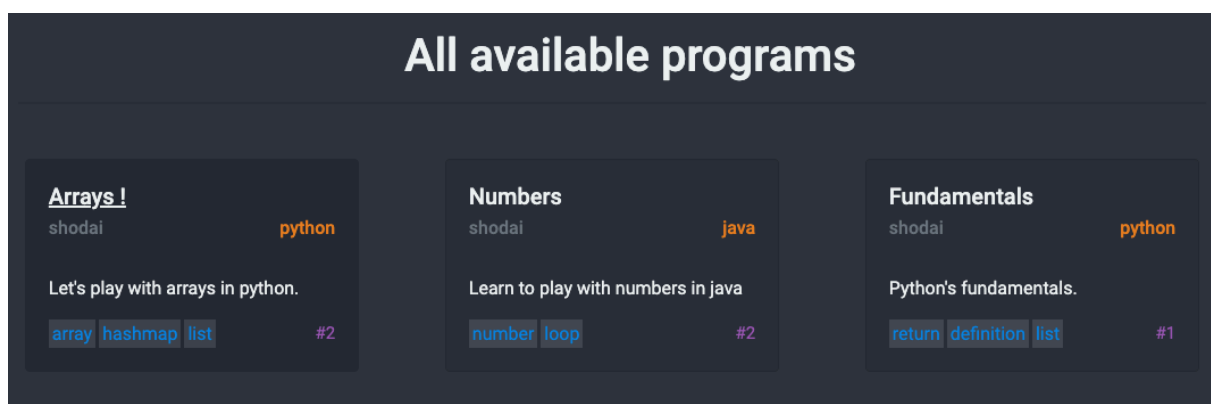


Figure 9 - Vue des programmes (AV, 2019)

Les programmes sont des ensembles de katas créés par un Sensei, auxquels un utilisateur peut s'abonner et y sauvegarder sa progression. Ils sont orientés sur un langage de programmation

en particulier, ce qui oblige son créateur à ne poster que des katas correspondants au langage du programme. De cette manière, le créateur est obligé de séparer les katas par langages et non par notions (voir Tableau 2).

Par notions	Par langage	
Programme - Les listes - Les listes en Java - Les listes en Python	Programme – Java - Les listes	Programme – Python - Les listes

Tableau 2 - Les programmes par notion / par langage

Les programmes sont agencés en “carte” (éléments de bootstrap, voir Figure 10). Cela oriente ainsi l'utilisateur de manière optimale lorsqu'il s'agit de trouver un programme

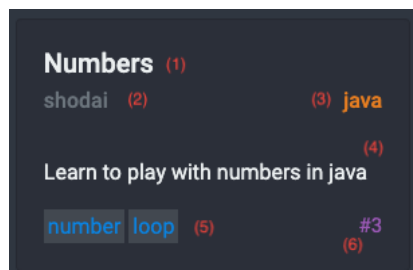


Figure 10 - Carte d'un programme (AV, 2019)

1. Titre
2. Nom du Sensei
3. Langage
4. Description
5. Tags
6. Nombre de kata

Filtrer un programme

Les cartes programmes ont été conçues pour être filtrables afin que l'utilisateur puisse s'y retrouver parmi la multitude de programmes présents sur la plateforme. En effet, il est possible d'appliquer un filtre (voir Figure 11 et Figure 12) sur le nom du Sensei, le langage du programme et ses tags.

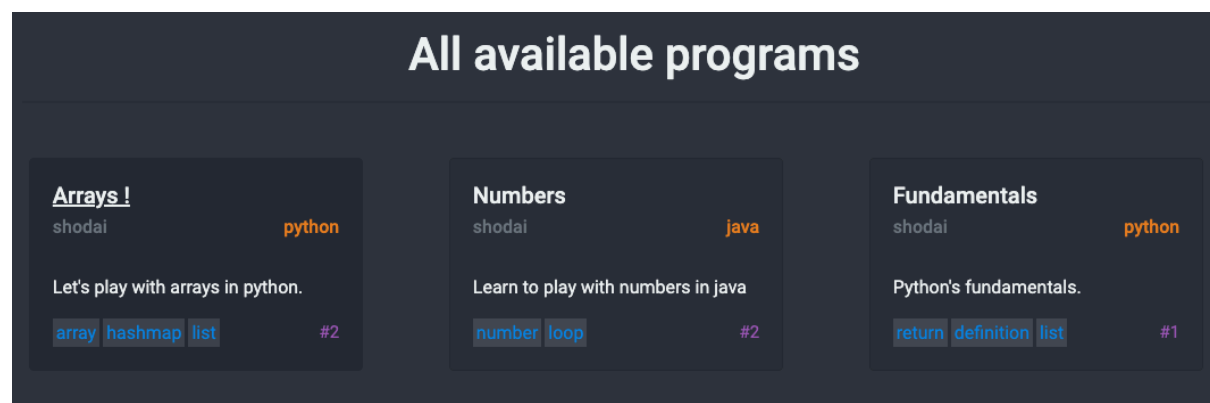


Figure 11 - Vue des programmes, aucun filtre (AV, 2019)

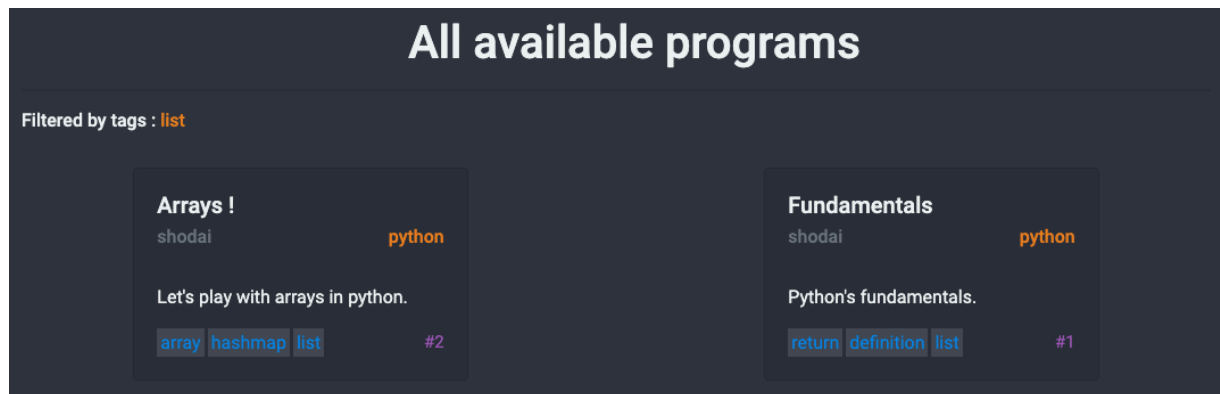


Figure 12 - Vue des programmes, Filtre sur tag "list" appliqué (AV, 2019)

Recherche

J'ai mis en place un outil de recherche pour que l'utilisateur puisse rechercher les programmes selon leur titre.

Gestion

Un Sensei se voit offrir un petit panel de gestion de programmes afin qu'il puisse les manipuler de manière optimale. Il peut ainsi créer son programme, le modifier, le supprimer, le dupliquer et même y appliquer un mot de passe. La duplication d'un programme implique évidemment de dupliquer tous ses katas, mais ne copiera pas la progression des utilisateurs abonnés

3.3.5 Les katas

Les katas (voir Figure 13) sont le centre névralgique de l'application ; c'est sur eux que s'est portée la plus grosse partie du développement. Ce sous-chapitre permet d'élaborer plus en détails l'utilité d'un kata, et les outils mis à disposition du Sensei pour les gérer.

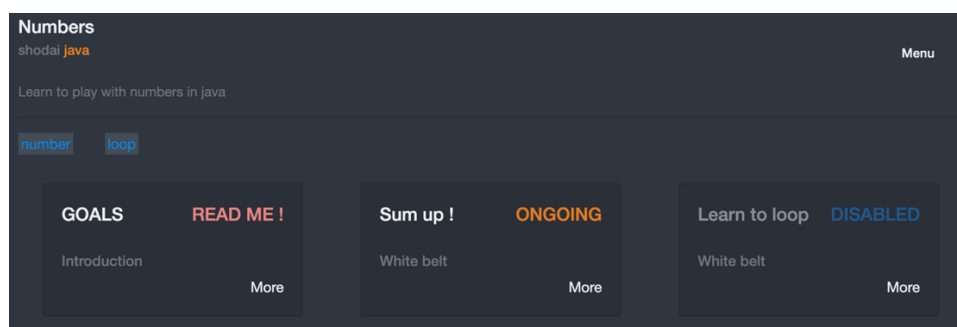


Figure 13 - Vue des katas, Sensei (AV, 2019)

Les katas sont des exercices de programmation créés par un Sensei, auxquels un utilisateur peut participer. En les réalisant, l'utilisateur peut faire évoluer sa progression dans un programme. Ils sont, eux aussi, agencés sous forme de carte (voir Figure 14), pour maintenir une cohérence dans l'affichage des ressources à travers la plateforme.

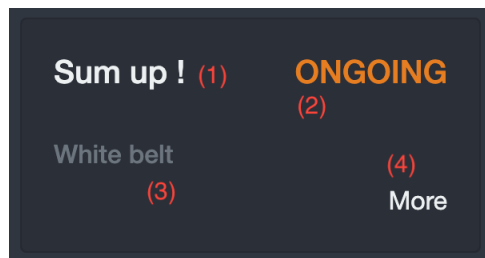


Figure 14 – Une carte kata (AV, 2019)

1. Titre
2. L'état
3. Niveau de difficulté
4. Menu du gestion, si authentifié en Sensei

Les états

Les états permettent de caractériser l'avancement d'un utilisateur sur un kata.

Nom	Action	Description
TODO	S'inscrire à un programme	L'utilisateur n'a pas encore cliqué sur la carte du kata en question
ONGOING	Cliquer sur la carte du kata en question.	Le kata est en cours de réalisation
RESOLVED	Résoudre le kata	L'utilisateur a réussi son kata, sa solution est enregistrée sur le serveur
FAILED	Abandonner le kata	L'utilisateur a abandonné le kata, il débloquent la solution
READ ME	S'inscrire à un programme	Le kata nommé "GOAL" (optionnel), doit-être lu, c'est les objectifs du programme

DONE	Cliquer sur la carte du GOAL en question.	L'utilisateur a pris connaissance des objectifs du programme
DISABLED	Le Sensei désactive le kata	Le kata n'est plus visible coté Monji
CLOSED	Le Sensei a fermé le kata	Le kata est visible coté Monji, mais n'est plus réalisable

Tableau 3 - Les différents status d'un kata

Interface de réalisation

Pour que l'utilisateur puisse réaliser un kata en ayant tous les outils en mains, j'ai dû imaginer une interface complète mais *simple* (l'analyse du marché ayant souligné l'importance de cette caractéristique). L'objectif était de guider naturellement l'utilisateur sur la page, la lecture se faisant de haut en bas, et de droite à gauche, toutes les informations devaient apparaître dans un ordre cohérent.

La Figure 15 représente l'interface de réalisation, avec un exercice de Java.

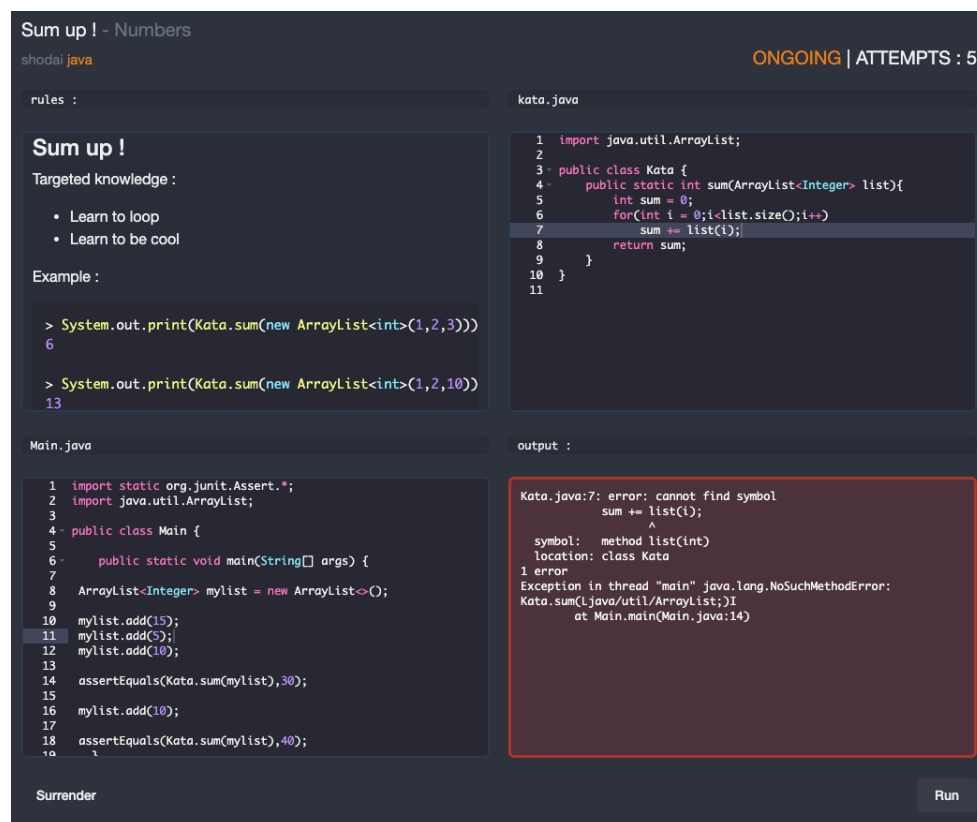


Figure 15 - Vue de la réalisation d'un kata (AV, 2019)

L'interface est découpée en 3 parties :

1. L'en-tête
2. Les outils de réalisation
3. Les actions

L'en-tête

L'en-tête contient un rappel contextuel du kata. A savoir : son titre, le programme auquel il appartient, le Sensei et le langage. De plus, il contient le statut actuel du kata et le nombre d'essais déjà effectués.

Les outils de réalisation

1. Règles du kata (rules, Figure 15)

Cette partie décrit l'objectif à réaliser pour le kata. Les règles peuvent-être écrites en texte pure, en markdown, mais peuvent aussi être une image (PNG, JPG) ou un PDF .

2. Canevas (*kata.java*, Figure 15)

Cette partie contient le canevas (bout de code incomplet) laissé par le Sensei lors de la création du kata. C'est ici que l'utilisateur devra écrire son code pour le faire valider.

3. Boîte de batterie de tests (*Main.java*, Figure 15)

La boîte de batterie de tests est une partie facultative de l'interface de réalisation d'un kata, le choix de la montrer ou non se fait dans la création d'un kata. Cette boîte affiche le code qui va évaluer la fonction que l'utilisateur a écrit.

4. Boîte de sortie (output, Figure 15)

C'est dans la boîte de sortie que va être montré le résultat de l'exécution du code écrit par l'utilisateur.

Ace editor

Les boîtes de canevas et de batterie de tests, découlent d'un module d'édition de code en temps réel, *ace editor* (ace.c9.io), ce module permet aussi la coloration syntaxique des langages. Il est donc très utile pour plonger l'utilisateur dans une bonne expérience d'outils de programmation en ligne.

Les options

J'ai décidé d'offrir à l'utilisateur le moyen d'abandonner un kata pour débloquer la solution au cas où il s'avérerait trop difficile pour lui. Toutefois, il lui faudra réaliser un certain nombre d'essais avant de pouvoir en débloquer sa solution. Le nombre d'essais est choisi par le Sensei lors de la création du kata. Le bouton « Run » (Exécution en français), permet de tester son raisonnement et de valider le kata s'il s'avère juste.

Si l'utilisateur abandonne ou réussit le kata, l'option « try again » (Refaire le kata), se débloque, et permet à l'utilisateur de refaire le kata, mais la réussite de ce nouvel essai ne changera plus le statut du kata.

Gestion

Comme pour les programmes, le Sensei se voit offrir de petits outils de gestion pour ses katas, pour pouvoir les manipuler de manière optimale. En effet, le Sensei peut modifier un kata ce qui remettra la progression à zéro du kata en question pour tous les utilisateurs abonnés au programme. De plus, il peut désactiver un programme, ce qui signifie qu'il ne sera plus visible du côté des abonnés. En outre, cette désactivation permet au Sensei de modifier son kata en toute sérénité, puisqu'aucun utilisateur ne pourra le réaliser pendant ce laps de temps. Un autre outil de gestion similaire à la désactivation est la fermeture du kata, cette fermeture laisse le kata visible du côté de l'abonné, pour qu'il puisse revenir sur l'exercice, mais il ne pourra plus le réaliser. Pour finir, le Sensei peut aussi supprimer un kata.

Créer un Kata

J'ai mis à disposition du Sensei, plusieurs outils pouvant créer un kata avec efficacité. Premièrement, le Sensei peut rédiger un ensemble d'instructions pour guider l'utilisateur à

travers son kata. Comme mentionné plus haut, ces instructions (ou objectifs) peuvent être écrits en texte pure ou en markdown. Mais encore, si le Sensei préfère, il peut poster les règles sous forme de document externe en tant que PDF (format préféré), JPEG ou PNG.

Pour ce qui est d'écrire l'exercice, le Sensei se voit offrir une interface similaire à celle de la réalisation d'un kata. En effet, l'interface est aussi séparée en 4 boîtes mais qui, ici, remplissent des rôles différents (voir Figure 16).

Note : L'interface de création d'un kata est plus amplement expliquée dans le manuel utilisateur.

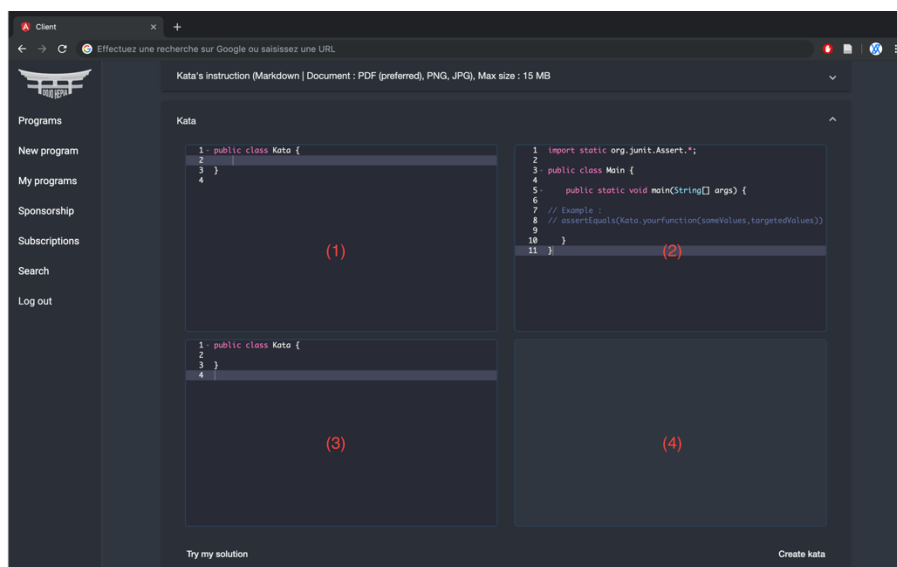


Figure 16 - Interface de création d'un kata (AV, 2019)

C'est dans la boîte (1) que le Sensei écrit sa solution. Dans la boîte (2), le Sensei rentre sa batterie de tests. Dans la boîte (3), le Sensei rentre le code minimum qui sera donné à l'utilisateur pour la réalisation du Kata. Pour finir, la boîte (4), affiche la sortie de l'exécution du code lorsque l'utilisateur le teste (à l'aide du bouton « try my solution »).

Pour guider un maximum le Sensei lors de son arrivée sur l'interface de création, des lignes de codes (voir Figure 16, dans les boîtes 1,2 et 3) sont déjà présentes (dépendamment du langage du programme, voir Tableau 4). Elles contiennent l'importation de la librairie d'assertion

associée au langage, un exemple d'utilisation de celle-ci, ainsi que le code minimal nécessaire pour que l'exécution soit satisfaite sans que l'utilisateur n'écrive un ligne.

Fichier de batterie de tests	Solution	Canevas
python		
<pre> from assertpy import assert_that from sample import * #Example #assert_that(urfunction(someValues)).is_equal_to(targetedValues) </pre>	<pre> # Write your code here </pre>	<pre> # Write your code here </pre>
java		
<pre> import static org.junit.Assert.*; public class Main { public static void main(String[] args) { // Example : //assertEquals(Kata.yourfunction(someValues,targetedValues)) } } </pre>	<pre> public class Kata { } </pre>	<pre> public class Kata { } </pre>

Tableau 4 - Le contenu des canevas, création d'un kata

Le fonctionnement de l'exécution du code est décrit dans le sous-chapitre « Le service de compilation ».

3.3.6 L'authentification

Pour rejoindre DojoHepia et s'y authentifier, j'ai imaginé un système basé sur un « passe ». En effet, aidé des JWT (qui sont détaillés dans la partie « Gestion des ressources et privilèges), je ne laisse rejoindre que les utilisateurs ayant reçu un passe.

Il existe deux types de passe :

- 1) Passe pour devenir Sensei
- 2) Passe pour devenir Monji

De cette manière, les Sensei peuvent aisément inviter leurs collègues à rejoindre la plateforme en tant que Sensei. Et inviter l'ensemble de leur classe sur la plateforme, avec un passe de type Monji.

3.4 LE SERVICE DE COMPILATION

Pour que ma plateforme puisse remplir sa fonction d'apprentissage et entraînement de *programmation en ligne*, j'ai dû réfléchir à une méthode qui me permettait de récupérer le code écrit par l'utilisateur, et de lui en ressortir le résultat de son exécution. Pour ce faire, je me suis d'abord orienté vers des outils d'exécution en ligne, tel que jdoodle (jdoodle.com). Malheureusement, les outils de ce type sont généralement payant et fonctionnent avec un système d'abonnement (ou alors contiennent des versions dites "d'évaluation", insuffisantes pour les besoins de DojoHepia). En effet, la nécessité de faire une plateforme propre à Hepia et qui ne dépend d'aucun service payant ou de licences non open-source, m'a poussé à m'orienter vers une solution "maison".

C'est de ce raisonnement qu'est née l'idée de créer mon propre service d'exécution, et la solution adoptée est d'exécuter le code que l'utilisateur souhaite faire valider sur un service dédié.

3.4.1 Mise en œuvre

Ce service est un serveur Javalin ayant pour objectif général d'exécuter et de renvoyer le résultat de l'exécution du code que l'utilisateur souhaite faire valider. Pour pouvoir tester le code de l'utilisateur, j'ai dû trouver une solution pour que le Sensei puisse lui-même créer une batterie de tests. La solution qui a tout de suite été retenue est d'utiliser les bibliothèques d'assertion car elles sont disponibles dans la plupart des langages et ne demande un niveau de connaissance que très peu approfondi.

Les assertions

Une assertion est une fonction de prédicat

$$pred(f(x), y)$$

qui permet de vérifier si le résultat de la fonction en entrée, x , correspond au résultat attendu, y . Cela permet au Sensei de créer une batterie de tests complète pour faire valider le code que l'utilisateur aura écrit. Voici les bibliothèques d'assertions utilisées pour l'implémentation des langages :

- Python : Assert Py (pypi.org/project/assertpy/)
- Java : Junit5 (junit.org/junit5/)

Assert Py

Si j'ai décidé de ne pas utiliser les assertions basiques qu'offre le langage python, c'est parce que cette librairie donne une description de l'erreur plus détaillée lorsque le prédicat s'avère faux. Voici un exemple de fonctionnement de cette librairie, avec différents tests.

Note : Dans ce kata, le but est de multiplier tous les éléments d'un tableau donné par un chiffre).

Sample.py (Solution)	main.py (Batterie de tests)
<pre>def mult(arr,n): return [i*n for i in arr]</pre>	<pre>from assertpy import assert_that import sample as m assert_that(m.mult([2,4,6],3)).is_equal_to([6,12,18])</pre>

Tableau 5 - Exemple d'utilisation d'assert py

Temps [s]	Résultat	Code d'erreur
Exemple : Kata réalisé avec succès		
1.99	Passed	0
Exemple : La fonction ne retourne pas le résultat attendu (assertion)		
1.75	AssertionError: Expected <[2, 4, 6]> to be equal to <[6, 12, 18]>, but was not.	1
Exemple : Erreur de syntaxe (générée par l'interpréteur Python)		
1.97	Traceback (most recent call last): File "assert.py", line 2, in <module> import sample as m File "/env/sample.py", line 2 return [i*n for i in arr ^ SyntaxError: unexpected EOF while parsing	1

Tableau 6 - Résultat de l'exécution d'un kata, assert py

Junit

Comme Java ne contient pas un système d'assertion de base, j'ai décidé d'utiliser Junit5, qui est une librairie complète de tests unitaire, Voici un exemple de fonctionnement de cette librairie, avec différents tests.

Attention : Dans ce kata, le but est de sommer tous les éléments d'un tableau donné.

Kata.java (Solution)	Main.java (Batterie de tests)
<pre>import java.util.*; public class Kata { public static int sum(List<Integer> list){ int sum = 0; for(int n : list) sum += n ; return sum ; } }</pre>	<pre>import static org.junit.Assert.*; import java.util.*; public class Main { public static void main(String[] args) { List<Integer> mylist = new ArrayList<>(Arrays.asList(15,5,10)); assertEquals(Kata.sum(mylist),30); mylist.add(10); assertEquals(Kata.sum(mylist),40) ; } }</pre>

Tableau 7 - Exemple d'utilisation de Junit 5

Exemple : Kata réalisé avec succès		
2.48	Passed	0
Exemple : La fonction ne retourne pas le résultat attendu (assertion)		
2.67	Exception in thread "main" java.lang.AssertionError: expected:<3> but was:<30> at org.junit.Assert.fail(Assert.java:93) at org.junit.Assert.failNotEquals(Assert.java:647) at org.junit.Assert.assertEquals(Assert.java:128) at org.junit.Assert.assertEquals(Assert.java:472) at org.junit.Assert.assertEquals(Assert.java:456) at Main.main(Main.java :9)	1
Exemple : Erreur de syntaxe (générée par le compilateur Java)		
2.34	Kata.java:8: error: ';' expected return sum ^	1

Tableau 8 - Résultat de l'exécution d'un kata, Junit 5

Avec ces librairies, il est donc possible de proposer au Sensei de composer un kata avec d'un côté le code qu'il souhaite faire réaliser, et de l'autre une batterie de tests qui permet de tester si le code réalisé fonctionne. L'exécuteur du kata aura accès aux erreurs générées par les

langages respectifs (à l'exécution), ainsi qu'aux erreurs liées à la batterie de tests, qui sont générées si la fonction rempli par l'utilisateur ne satisfait pas le prédicat.

3.4.2 Containerisation du code

J'ai décidé d'isoler l'exécution du code dans un container à l'aide de Docker (docker est un utilitaire de création de container, de petites machines virtuelles fonctionnant sur le kernel hôte), pour éviter des problèmes de sécurité évidents liés à une exécution locale, tel que la suppression de fichier, l'accès aux ressources non légitimé, la modification, etc...

Pour démontrer l'utilité de containeriser l'exécution du code, voici un tableau de comparaisons entre les deux méthodes :

Tableau de comparaison – Exécution Hôte vs Container			
Exécution sur l'hôte		Exécution containerisée	
Avantages	Inconvénients	Avantages	Inconvénients
Plus rapide	Peu modulaire	Modulaire	Plus lent
Facilité de préparer un service contenant tous les besoins	Peu ou pas de sécurité	L'exécution se faisant dans un container, l'utilisateur n'a accès qu'à son contenu	
	Peu portable sur des plateformes de déploiement	Facilité de créer un container par implémentation de langage	
		La maintenance d'une image simple + modification aisée	
		Comme le container est supprimé après son utilisation, y effacer des fichiers ou corrompre son contenu ne posera pas de problèmes	

Tableau 9 - Avantages et inconvénients de la containerisation de l'exécution.

Comme le montre le Tableau 9, la containerisation de l'exécution est largement plus profitable que l'exécution sur le serveur hôte. La seule chose qui pourrait poser problème est le temps d'exécution (voir Tableau 10), mais c'est un mince sacrifice pour les avantages que nous apportent les containers

Note : Tests réalisés sur un Macbook Pro 2015 (Tests réalisés 10x)

250 Go de stockage, 8 Go de ram, Intel Core i5 (2,7 GHz), processeur déjà en charge

Temps [s]	Python	Java
Hosted – Moyenne	0,035	1,534
Container – Moyenne	1,56	2.276

Tableau 10 - Mesures, Exécution containerisée vs hostée

Un des points importants relevés dans les avantages de la containerisation (voir Tableau 9), c'est que l'on peut créer une image pour des besoins bien spécifiques. En effet, dans le cadre de DojoHepia, j'ai dû créer deux images qui contenaient les outils nécessaires à l'exécution des langages Java et Python.

3.4.3 Les images

L'image Python

```
FROM python
RUN pip3 install assertpy
RUN mkdir /env/
```

Listing 5 - Dockerfile de l'image python

Comme le montre le Dockerfile (fichier permettant de construire une image, voir docs.docker.com/engine/reference/builder/) du Listing 5, l'image python est basée sur 'python' qui est une image déjà présente sur Dockerhub (hub.docker.io), Cela permet d'avoir une image python stable et complète. Il suffit juste d'ajouter la librairie d'assertion. La commande « RUN mkdir /env/ » crée un répertoire à la racine de l'image nommée « env », c'est ce répertoire qui sera attaché au répertoire partagé lors de la création du container (voir section « Répertoire partagé, attaché au container »).

L'image Java

```
FROM ubuntu:14.04

RUN apt-get update && apt-get -y upgrade
RUN apt-get install -y default-jdk

RUN mkdir /env/
ADD .bashrc /root/
```

Listing 6 - Dockerfile de l'image java

Cette fois-ci, le Dockerfile (voir Listing 6), est un peu plus complexe. Premièrement, l'image ne se base pas sur des images comme « openjdk » ou « java » déjà existantes sur le dockerhub. En effet, l'image se base sur l'image Ubuntu 14.04, qui me permet d'installer sans trop de complexité le jdk d'oracle qui, lui, me permet de compiler et exécuter le langage java. Il se charge aussi d'ajouter le fichier `.bashrc` (voir Listing 7), qui permettra de référencer le jar de la librairie d'assertion (JUnit5) plus tard, lors de la création du container.

```
export CLASSPATH=junit-4.10.jar:.
```

Listing 7 - Contenu du fichier .bashrc

Le répertoire partagé (voir aussi plus bas, dans la partie « Répertoire partagé, attaché au container ») sur la machine hôte contient déjà le fichier, « `java_test.sh` », qui est une petite suite de commandes bash (voir Listing 8), permettant d'effectuer des actions utiles avant de pouvoir correctement utiliser le container, notamment le référencement du `.jar`, ainsi que la compilation des deux fichiers, `Main.java` et `Kata.java`. De plus, elle contient le `.jar` de Junit5.

```
#!/bin/bash
source /root/.bashrc
javac -classpath ${CLASSPATH} Main.java Kata.java # Compile files and librairies
java Main # Run the application
```

Listing 8 - Contenu du fichier java_test.sh

Répertoire partagé, attaché au container

Pour que les containers puissent utiliser les ressources générées par le service de compilation, celui-ci les stocke dans un répertoire partagé, attaché au container lors de son démarrage. En effet, cela permet au container d'avoir accès aux fichiers nécessaires sans devoir générer une image à chaque nouvelle exécution. L'utilisation de volumes partagés n'est pas nécessaire car

leurs avantages ne profitent pas à mon implémentation (pour de plus amples détails sur les volumes, voir : docs.docker.com/storage/volumes/).

Lancement des containers, docker run

Voici une description des deux commandes permettant de créer et lancer les containers Python et Java

Python

```
docker run --rm --mount type=bind,source=AbsolutePath/share_docker_file,dst=/env/  
freakency/python:3.0 python assert.py
```

Listing 9 - Commande docker, permettant de créer et lancer un container python

Java

```
docker run --rm --mount type=bind,source=AbsolutePath/share_docker_file,dst=/env/  
freakency/java:1.0 ./java_test.sh
```

Listing 10 - Commande docker, permettant de créer et lancer un container java

Options de la commande docker run :

-- rm : Permet de supprimer automatiquement le container après la fin de son exécution.

-- mount : Permet d'attacher le répertoire partagé au container.

-- [..] : Permet d'indiquer sur quelle image est basé la création du container.

-- [..] : Permet d'indiquer une options de lancement, pour python (voir Listing 9), lance l'interprétation du fichier « assert.py », pour java (voir Listing 10) lance l'exécution de « java_test.sh ».

Pour de plus amples informations sur les options de la commande docker run, veuillez-vous rendre sur :

docs.docker.com/engine/reference/run/ (Docker run - Documentation 2019)

Figure 17 présente le flux de communication entre les différents services concernant l'exécution d'un kata.

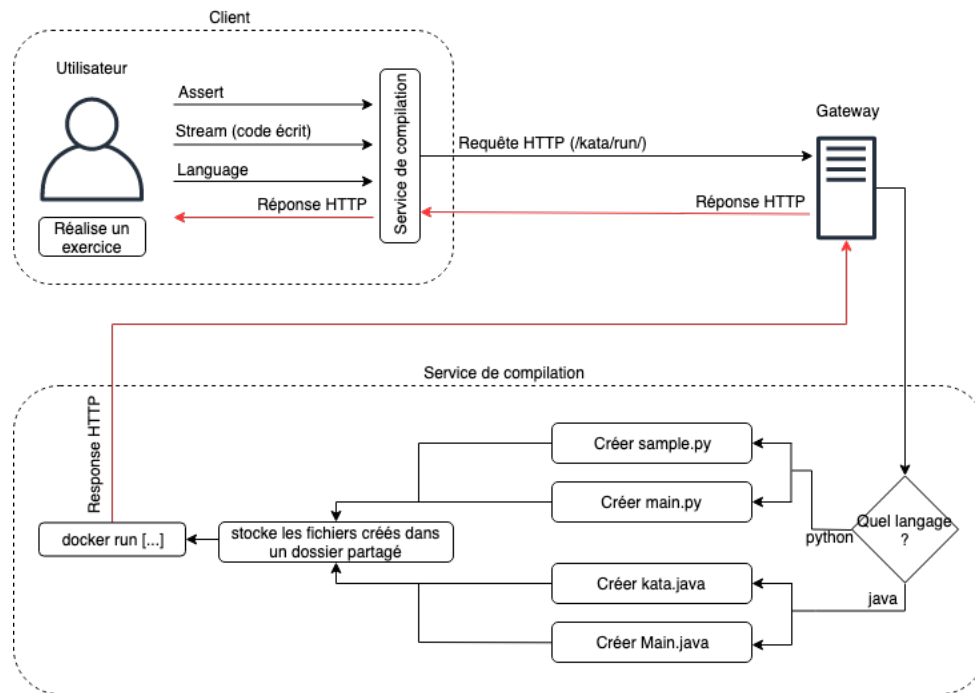


Figure 17 – Réalisation d'un kata, communication entre les services (AV, 2019)

Description

Comme le montre la

Figure 17, le Client émet en direction du Gateway un appel API (à /kata/run, voir Tableau 11) qui permet d'envoyer le code pour exécution. Ce code reçu par le Gateway, est envoyé au service de compilation qui traitera l'information et exécutera son workflow pour répondre au Gateway (voir Tableau 12).

Requête HTTP, Client -> Gateway -> Service de compilation		
Nom du champ	Type	Contenu
stream	string	Contenu du code de l'utilisateur
assert	string	Batterie de tests

language	string	Langage cible
----------	--------	---------------

Tableau 11 - Contenu du body, Client -> Gateway -> Service de compilation

Requête HTTP, Service de compilation -> Gateway -> Client		
Nom du champ	Type	Contenu
output	String	Contenu de la sortie de l'exécution si le code s'est exécuté avec succès
error	String	Contenu de la sortie de l'exécution si le code n'a pas pu s'exécuter correctement
exit	Integer	Code d'erreur après exécution
time	long	Temps de l'exécution

Tableau 12 - Contenu du body, Service de compilation -> Gateway -> Client

3.5 GESTION DES RESSOURCES ET PRIVILÈGES

Comme vu dans le chapitre “Conception - Architecture”, j’ai choisi d’implémenter un service de type sans état (stateless en anglais) pour le Gateway. Il a donc fallu trouver un moyen d’avoir des utilisateurs authentifiés, sans que le serveur n’en dépende. Pour ce faire, j’ai mis en place un système d’authentification avec JWT.

3.5.1 Json Web Token

Explications tirées du site officiel de JWT (jwt.io/introduction/). Le but de cette introduction est de faire l’analogie avec mon implémentation.

JWT, ou “JSON Web Token” est un standard d’échange d’informations sécurisé au format JSON. Les informations transmises avec JWT sont fiables et ce, parce qu’elles peuvent être signées avec un secret ou une paire clé privée / clé public (RSA ou ECDSA). J’utilise pour ma part un secret pour générer les tokens.

Un JWT a trois composantes :

- Header
- Payload
- Signature

et tout assemblé, ressemble à ca :

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1NTkwNDM4MTh9.EiUc5iYOzw8C6JLGUts7Eemm2C7pZhrrEpEhmYGbWjA
```

Listing 11 - Un JWT

Header

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
```

Le header (voir Listing 12) est généralement composé du type (pour moi, JWT) de token, et de l’algorithme de hachage utilisé (qui est HMAC256 dans mon cas).

```
{
  "alg": "HMAC256",
  "typ": "JWT"
}
```

Listing 12 - Header d’un JWT

Payload

```
eyJleHAiOjE1NTkwNDM4MTh9.
```

Le payload (voir Listing 13) est quant à lui composé de “Claims”, qui sont des informations concernant l’utilisateur de manière générale. Dans mon implémentation, lors de la génération du token le payload contient :

Contenu du payload – JSON Web Token	
Nom du champ	Description
username	Le pseudo de l’utilisateur
level	Le niveau de privilège de l’utilisateur
exp	La date (à la milliseconde près) à laquelle le token a été généré.

Tableau 13 - Payload d'un JWT

Le champ “username” permet d’avoir une composante unique (étant donné qu’un nom d’utilisateur est unique), et de ce fait, avoir un token différent au moins pour chaque utilisateur, le champ “level”, permet au Gateway de gérer l’accès aux ressources (un utilisateur avec le rang “SHODAI” n’aura pas les mêmes droits qu’un utilisateur avec le rang “MONJI”), et le champ “exp”, permet de donner une date d’expiration, après laquelle, le token ne sera plus valide. Voici un exemple :

```
{
  "username" : "alice",
  "level" : "MONJI",
  "exp" : 1559046330
}
```

Listing 13 - Exemple du payload d'un JWT

Signature

EiUc5iYOzw8C6JLGUs7Eemm2C7pZhrrEpEhmYGbWjA

La signature est un hash composé du header, du payload, et, pour mon implémentation, d’un secret (un secret est une donnée de type texte, contenant une valeur choisie, et qui doit rester secrète). Ce hash est généré grâce à l’algorithme spécifié dans le header.

Signature : HMAC256(base64UrlEncode(**header**) + « . » + base64UrlEncode(**payload**),
secret)

3.5.2 Mise en œuvre

Le Client et le Gateway sont dotés d’un système de rôles, qui permet de définir un niveau de hiérarchie pour l’accès aux ressources. Ce niveau de hiérarchie se découpe de la manière suivante (du rang le plus influent, au moins influent) :

- SHODAI
- SENSEI

- MONJI
- ANYONE (*tout le monde*)

Ces niveaux permettent de définir qui pourra accéder à quelle route, côté Gateway, et qui pourra accéder à quelle page, côté Client. Une requête émise en direction du Gateway doit toujours contenir le token fournit lors de l'authentification, à deux exceptions près : la route de création de compte (/user/signin) et la route d'authentification (/user/tokenrequest/), qui sont des routes dotées d'un niveau d'accès ANYONE (n'importe qui peut y accéder).

Demande de token

Lorsqu'un utilisateur souhaite s'authentifier pour accéder à ses ressources, il effectue une demande de token au Gateway, cette demande peut se dérouler de deux manières :

1. Les données d'authentifications ne sont pas valides
2. Les données d'authentifications sont valides

Si l'utilisateur fournit des données d'authentification correctes (qui est une paire *<nomUtilisateur,motDePasse>* correcte), le serveur lui fournira un Token d'accès aux ressources en plus de son nom d'utilisateur, son id d'utilisateur et de son rôle. Ces informations, une fois reçues, sont stockées par le Client dans le LocalStorage. Le LocalStorage permet de stocker de manière persistante et locale des données sous une forme *<clé ;valeur>*, et ce, même si l'utilisateur ferme son navigateur.

Accès à une ressource

Lorsqu'un utilisateur souhaite accéder à une ressource, le Client va émettre une requête HTTP en direction du Gateway. Comme spécifié plus haut, chaque requête émise doit être accompagnée d'un Token. Pour ce faire, j'ai mis en place un service d'interception de requêtes HTTP.

La Figure 18 décrit le cycle de vie d'une requête HTTP, qui est détaillé dans les prochaines pages.

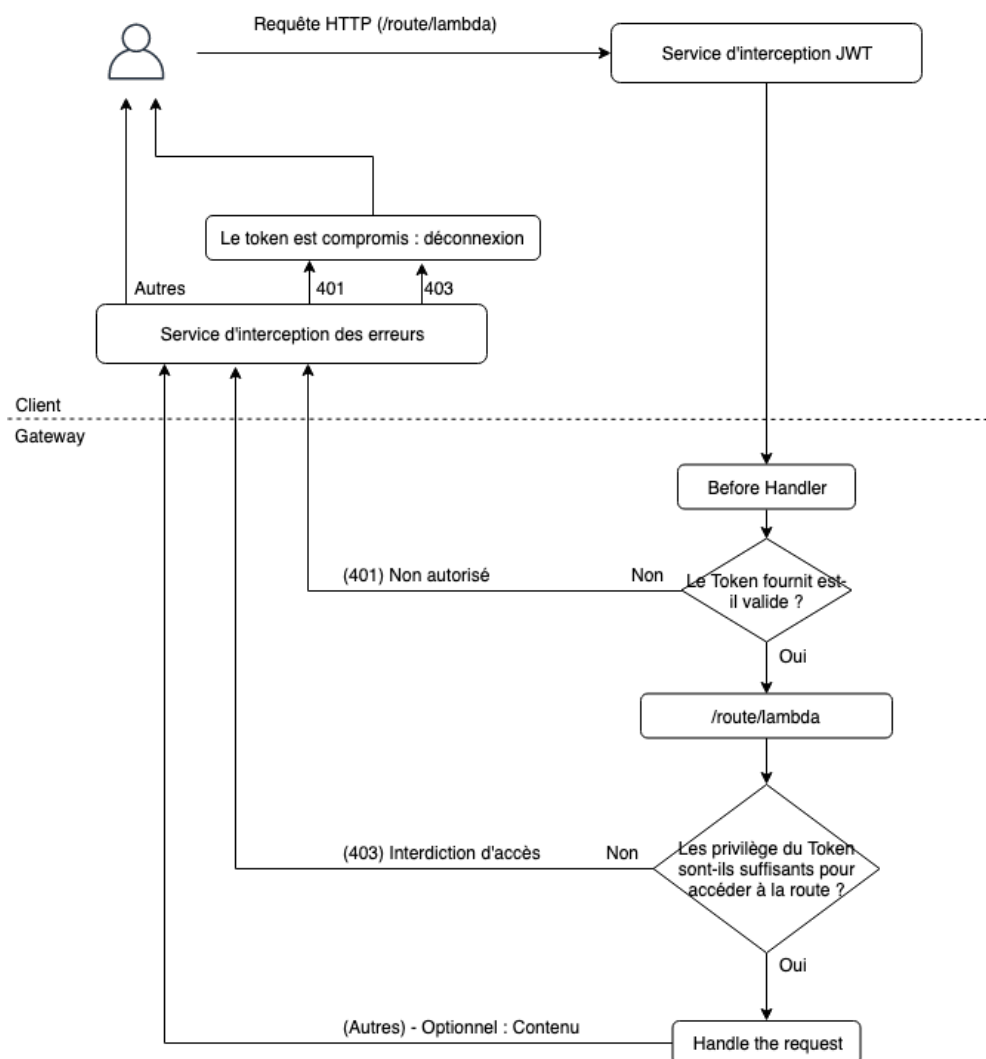


Figure 18 - Cycle de vie d'une requête HTTP (AV, 2019)

Angular Services – JWT Interceptor

Ce service permet d'intercepter chaque requête émise, et d'y coller le token stocké dans le LocalStorage dans les en-têtes. Ainsi, cela me permet de ne pas surcharger tous les autres services. A partir de là, le serveur peut répondre de trois manières différentes :

Code de la réponse	Description
Autres	Succès – Optionnel : Contenu
401	Non autorisé
403	Interdiction d'accès

Tableau 14 - Réponse du Gateway, à l'accès d'une ressource

Angular Services – Error Interceptor

J’ai aussi mis en place un service d’interception des erreurs, ce service colle un objet de type “Observable” (angular.io/guide/observables) aux requêtes émises, et lorsque celles-ci reçoivent une réponse du serveur, le code d’erreur reçu est intercepté par ce service. Cela permet de centraliser tous les retours de requêtes en provenance du Gateway, et de pouvoir gérer les codes d’erreurs 401 et 403. Si le serveur renvoie un de ces deux codes, il détruit l’utilisateur du LocalStorage ce qui compromet son Token, et fait office de déconnexion.

En ce qui concerne le Gateway, chaque route est précédée de ce qu’on appelle un “before handler” (un petit bout de code qui s’exécute avant le code principal de la route). Ce handler s’assure que l’entête de la requête contient bien un Token valide. La validité d’un Token dépend de plusieurs facteurs :

1. Le Token est-il contenu dans les en-têtes HTTP ?
2. La Date d’expiration du Token excède-t-elle la date de réception de la requête ?
3. Le Token est-il décodable avec la clé secrète ? (Intégrité du token)

Token non valide

Le Gateway renvoie un code d’erreur (401) « Non autorisé », ce qui signifie que l’utilisateur n’est pas authentifié et qu’il doit l’être pour continuer. Si le token est valide mais que le rang contenu dans les “Claims” du token ne lui permet pas d’accéder à la route désirée, le Gateway renverra le code d’erreur (403) « Interdiction d’accès », ce qui signifie que l’utilisateur n’a pas les droits requis pour accéder à la ressource (Dans mon cas, une erreur de type 403 ne devrait pas arriver si l’utilisateur utilise le site de manière conventionnelle car les accès sont aussi gérés côté Client. De ce fait, lors d’une erreur 403 l’utilisateur est intentionnellement déconnecté du service).

Token valide

Le serveur traite la requête, et renvoie le contenu avec un code de réponse (200).

Accès aux pages du côté Client

Lorsqu'un utilisateur souhaite accéder à une route un "Authentication Guard" (qu'on pourrait apparenter au *before handler* du Gateway) regarde si l'utilisateur remplit les conditions nécessaires pour y accéder. Il y a deux conditions :

1. Être connecté
2. Avoir le rôle requis pour accéder à la route

Si la première condition n'est pas respectée, l'utilisateur est automatiquement renvoyé sur la page de connexion. Quant à la deuxième condition, si elle n'est pas respectée l'utilisateur est renvoyé à la page d'accueil.

Cette implémentation permet de s'éviter des problèmes ingérables d'accès aux ressources non autorisées (ex : l'utilisateur accède à une route qui lui est restreinte et envoie une requête avec des privilèges insuffisants au serveur).

CONCLUSION

DojoHepia est la solution idéale pour orienter Hepia vers une réforme de ses outils scolaires afin d'évoluer avec son temps et de tendre vers l'enseignement 2.0.

L'étude de marché effectuée a permis de mettre en avant les différents avantages et inconvénients de plateformes similaires et a mis en lumière ce que DojoHepia ne devait pas reproduire. Cela a créé le terrain pour que ma plateforme se construise sur de bonnes bases. A l'inverse des autres options sur le marché, les fonctionnalités ne se sont pas développées au dépend de qualité design et d'expérience utilisateur. J'ai aspiré à trouver un équilibre entre complexité et harmonie pour rassembler tous les éléments importants pour une application web.

Grâce au système de programmes composés de katas, le client peut vivre une expérience ludique et complète au travers d'une interface accessible. L'outil de programmation étant en ligne, toutes les étapes de l'apprentissage se retrouvent au même endroit ce qui facilite un suivi du développement éducatif. Cela a été rendu possible par la mise en place d'un service de compilation, d'un Gateway et d'une base de données. Cette architecture permettra des améliorations de la plateforme et de d'ores et déjà palier à des problèmes de montée en charge à l'aide d'un potentiel déploiement cloud.

Certains objectifs que Mr. Cavat et moi-même s'étions fixés n'ont pas été réalisés car la méthodologie de travail adoptée nous a dirigé à préférer certaines tâches plutôt que d'autres, permettant ainsi de maximiser la valeur ajoutée de chaque nouvelle itération. Néanmoins, ce travail et ce qu'il représente prouvent que les objectifs du cahier des charges ont largement été couverts et que DojoHepia se dote d'encore plus de fonctionnalités que prévu. En effet, au fil des semaines et du développement, de nouvelles idées d'amélioration sont apparues et c'est souvent elles qui ont été mises en priorité. L'étude de la gamification qui n'aura malheureusement pas été mise en œuvre, aura quand même permis de mettre en avant le besoin de cette fonctionnalité et de ses mécanismes.

Je n'avais jamais eu la chance de créer un projet autant conséquent avec une méthodologie de travail se rapprochant des standards industriels. J'ai pu comprendre que certains choix de

fonctionnalités ont vraiment plus de valeur que d'autres, et qu'une application n'est jamais réellement finie car il y a toujours matière à amélioration.

Ce travail m'aura marqué notamment de par mes découvertes de nombreuses technologies comme Angular CLI ou encore Javalin, qui sont des outils puissants et efficaces que je ne manquerai pas de réutiliser. Il m'a aussi permis de me perfectionner sur les notions abordées au cours de mes trois ans d'apprentissage à Hepia que ce soit le développement web, la virtualisation, ou encore l'implémentation du Gateway en java qui m'a enseigné quelles étaient les bonnes pratiques et conventions pour qu'une architecture REST ou qu'une application Java soit robuste et fiable.

Cette conclusion n'est pas une fin en soi pour DojoHepia. En effet, la plateforme doit encore s'acquitter de fonctionnalités importantes pour rendre l'expérience plus immersive. L'amélioration la plus pertinente qui devra être apportée est le système de worker dont j'ai brièvement parlé dans la partie « Architecture – Vision à long terme ». J'aurais aussi aimé mettre en place un thème clair pour que l'utilisateur puisse customiser un peu la plateforme. En effet, chez nous jeunes développeurs, la couleur de notre environnement de travail est souvent sujette à débat, et pouvoir s'affranchir des deux est devenu un must have pour des applications de développement. Pour continuer, un système d'administration complet serait nécessaire pour que le site ne devienne pas un lieu pourri par les utilisateurs. Ce système pourrait même s'étendre jusqu'à la gestion de groupe d'utilisateurs abonnés à un programme par le Sensei. Un système d'issues/commentaires à la manière de Codewars ne serait pas à négliger pour que l'utilisateur puisse donner son avis sur un programme, un kata et sur la plateforme de manière générale. Pour finir, un déploiement de la plateforme sur mobile serait un point intéressant à étudier. Non seulement cela permettrait de créer de petits exercices à réaliser à la minute, mais aussi de visualiser les informations de son compte, des statistiques ou commentaires pour un suivi plus confortable qu'en sortant son ordinateur.

ANNEXES

ANNEXE 1 : MANUEL DE L'UTILISATEUR

SENSEI

Attention : Dans cette partie, seule les fonctionnalités qui ne sont pas communes au Monji seront détaillées, si vous souhaitez en apprendre plus sur elles, rendez-vous dans la section « Monji »

En tant que Sensei, vous avez la possibilité de :

- Créer un programme
- Gérer votre programme
 - Suppression
 - Edition
 - Duplication
- Créer un kata
- Gérer votre kata
 - Suppression
 - Edition
 - Désactivation
 - Fermeture
- Créer un goal
- Visualiser vos programmes
- Sponsoriser un nouvel utilisateur
 - Monji
 - Sensei



Figure 19 - Manuel utilisateur : N'importe qui - Carte d'un programme (AV, 2019)

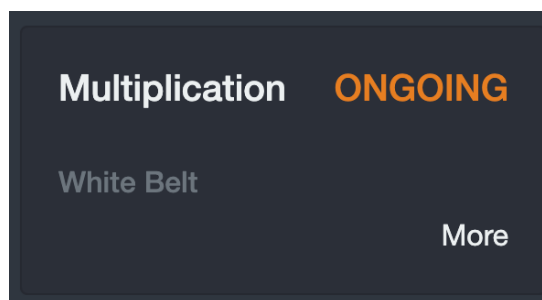


Figure 20 - Manuel utilisateur : N'importe qui - Carte d'un kata (AV, 2019)

Créer un programme

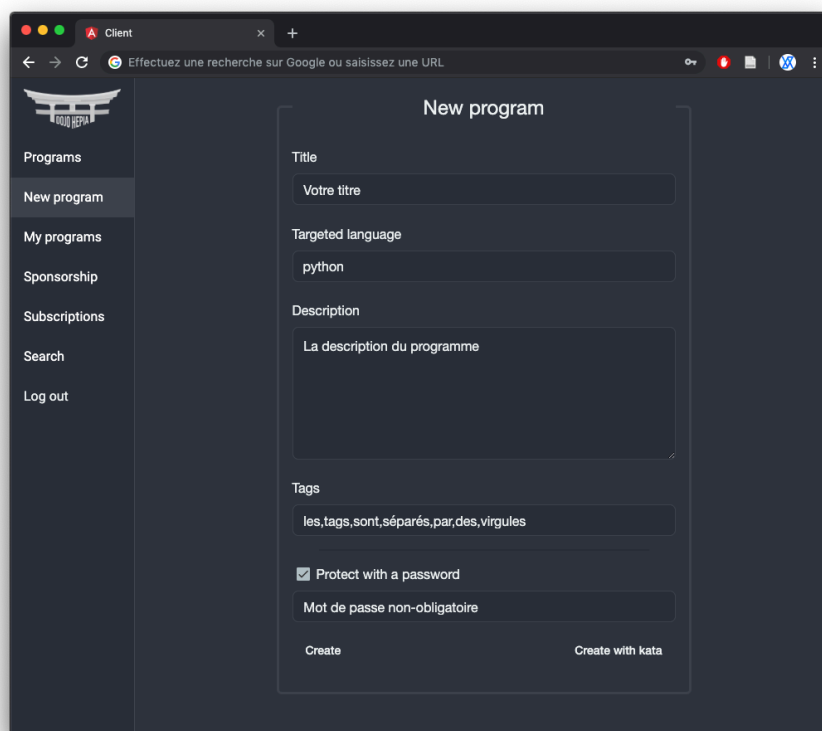
The screenshot shows a web browser window with the title 'Client'. The address bar contains the text 'Effectuez une recherche sur Google ou saisissez une URL'. The browser's tabs and extensions are visible. The application's sidebar on the left contains a logo at the top, followed by a list of menu items: 'Programs', 'New program' (which is highlighted), 'My programs', 'Sponsorship', 'Subscriptions', 'Search', and 'Log out'. The main content area is titled 'New program' and contains several input fields: 'Title' with the placeholder 'Votre titre', 'Targeted language' with the placeholder 'python', and 'Description' with the placeholder 'La description du programme'. Below these is a 'Tags' field with the placeholder 'les, tags, sont, séparés, par, des, virgules'. There is a checkbox labeled 'Protect with a password' which is checked, followed by a 'Mot de passe non-obligatoire' field. At the bottom of the form are two buttons: 'Create' and 'Create with kata'.

Figure 21 - Manuel utilisateur : Sensei - Page d'un programme (AV, 2019)

Voici le menu de création d'un programme. Vous devrez y rentrer son titre, le langage souhaité (python, ou java), sa descriptions ainsi que des tags. Les tags sont primordiaux car ils permettent d'orienter l'utilisateur et de décrire votre programme de manière très intuitive. Vous devez séparer vos tags d'une virgule (« , ») pour qu'ils s'affichent correctement sur la carte d'un programme. De plus, vous pouvez renseigner un mot de passe (non obligatoire), qu'un utilisateur devra donner lors de son abonnement au programme. L'option « create » (créer, en français), créer le programme, et « create with kata » (introduire un kata à la création, en français) vous amène sur la page de réalisation d'un kata.

Gérer votre programme

DojoHepia met à votre disposition 3 outils de gestion, qui vous permettent de manipuler vos programmes. Le menu de gestion (voir Figure 22) se trouve dans la page d'affichage d'un programme, en haut à droite.

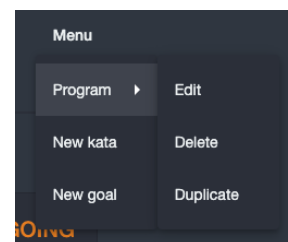


Figure 22 - Manuel utilisateur : Sensei - Menu de gestion d'un programme (AV, 2019)

Suppression d'un programme

Supprimer un programme est définitif. La suppression d'un programme entraîne l'effacement des katas du programme et la progression des utilisateurs abonnés.

Edition d'un programme

Éditer un programme vous donne la possibilité de modifier tous les champs renseignés dans la partie « Créer un programme ».

Note : Vous ne pouvez pas changer le langage cible d'un programme

Duplication d'un programme

Dupliquer un programme copiera tous les katas de celui-ci, mais ne copiera pas la progression des utilisateurs. Cela vous permet de le mettre à jours sans impacter la progression d'un utilisateur. Lors d'une duplication, vous devez renseigner un nouveau nom de programme.

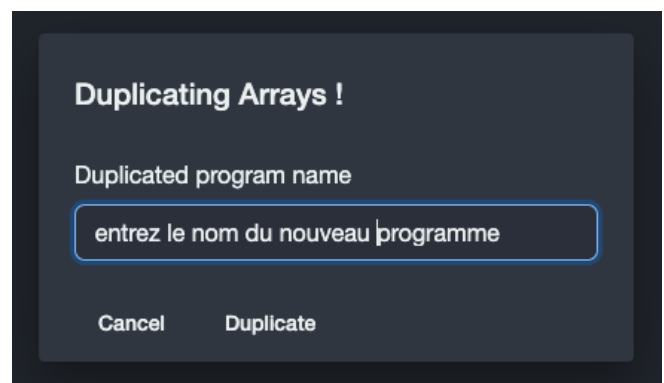


Figure 23 - Manuel utilisateur : Sensei - Menu de duplication d'un programme (AV, 2019)

Créer un kata

Pour créer un kata, rendez-vous dans le menu de gestion d'un programme (voir partie « Gérer votre programme ») : menu -> new kata (nouveau kata, en français). Vous êtes alors sur la page de création d'un kata comme sur la Figure 24. La création d'un kata est séparée en 3 parties :

1. Kata's information, qui regroupe les informations primaires d'un kata (titre, voir la batterie test, nombre d'essais avant déblocage de la solution)
2. Kata's instruction, les règles de votre kata. Vous pouvez rédiger les règles en markdown

Aide pour le markdown :

github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet (Pritchard 2019).

Ou y insérer un document. Les document peuvent-être de nature PDF, JPEG ou PNG et ne doivent pas dépasser la taille maximale de 15 Mb.

Attention : Vous ne pouvez pas modifier le document d'un kata après création de celui-ci.

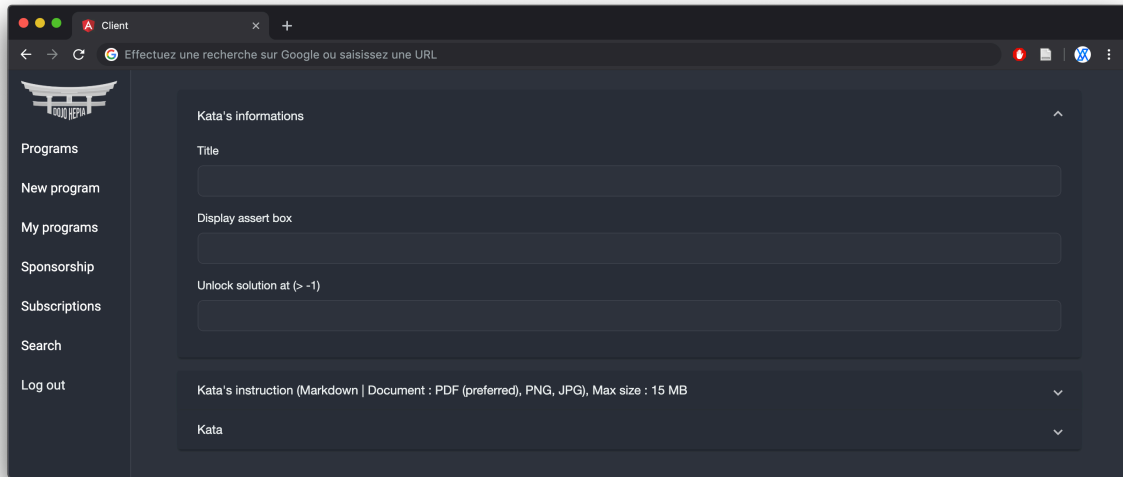


Figure 24 - Manuel utilisateur : Sensei - Menu de création d'un kata (AV, 2019)

3. Le Kata

C'est ici que vous allez insérer le code de votre kata. La partie 3 se sépare en 4 boîtes (Visible sur la Figure 24).

- (1) Cette boîte est le code contenant votre solution finale
- (2) Cette boîte est votre batterie de tests, les fonctions de la boîte (1) sont toutes disponibles du moment que vous laissez la ligne « `from sample import *` » (pour python. Pour java, la classe de la boîte (1) est compilée avec le fichier Main).

Les batteries de tests fonctionnent généralement avec des librairies d'assertions, qui vous sont proposées nativement.

Pour python : Assertpy

pypi.org/project/assertpy/

Pour java : Junit5

junit.org/junit5/docs/current/user-guide/#writing-tests

Vous avez la possibilité de créer vos propres fonctions de tests, mais il vous est fortement conseillé d'utiliser les bibliothèques de batteries de tests proposées. La boîte (2) est un exemple d'utilisation de la bibliothèque *assertpy*.



Figure 25 - Manuel utilisateur : Sensei - Création d'un kata (AV, 2019)

(3) Est le code canevas que vous laissez à vos utilisateurs (voir partie « réaliser un kata »).

(4) Est la boîte de sortie de votre kata, lorsque vous souhaitez le tester.

Le bouton « try my solution » (Essayer ma solution, en français) permet de savoir si le code écrit en (1) passe les tests écrits en (2). Et le bouton « create kata » (Créer le kata) permet d'ajouter le kata à votre programme.

Gérer votre kata

DojoHepia met à votre disposition 4 outils de gestion, qui vous permettent de manipuler vos katas. Le menu de gestion (voir Figure 26) est accessible au clic du bouton « more » (plus, en français) sur une carte kata.

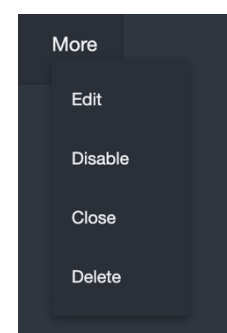


Figure 26 - Manuel utilisateur : Sensei - Menu de gestion d'un kata (AV, 2019)

Suppression d'un kata

Supprimer un kata est une action définitive. La suppression d'un kata entraîne l'effacement de la progression des utilisateurs abonnés au kata en question.

Edition d'un kata

Éditer un kata vous donne la possibilité de modifier tous les champs renseignés dans la partie « Créer un kata ».

Attention : Vous ne pouvez pas modifier le document d'un kata (si vous avez créé un kata avec document).

Attention : L'édition d'un kata entraîne la suppression de la progression des utilisateurs abonnés sur le kata en question.

Désactiver un kata

Un kata désactivé n'est pas disponible pour les utilisateurs, mais vous pouvez toutefois le modifier pendant sa désactivation.

Attention : Un kata ne peut être désactivé si il est fermé

Fermeture d'un programme

Un kata fermé est visible mais n'est plus réalisable.

Attention : Un kata ne peut être fermé si il est désactivé.

Créer un goal

Un goal est l'objectif général d'un programme, vous pouvez en créer le nombre que vous souhaitez, mais il est toutefois conseillé de n'en créer qu'un. Un goal est forcément rédigé en markdown .

Aide pour le markdown :

github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet (Pritchard 2019).

Visualiser vos programmes

La page « My programs » (Mes programmes, en français), permet de visualiser les programmes que vous avez créé.

Sponsoriser un nouvel utilisateur

Pour que quelqu'un puisse rejoindre DojoHepia, il doit référencer un token lors de la création de son compte. C'est ce que DojoHepia appelle « Le sponsoring ». Sur la page de sponsoring, vous avez le choix de générer un token pour un Monji ou un Sensei, et de choisir le temps avant que le token n'expire (vous pouvez voir les tokens comme des cartes d'invitation).

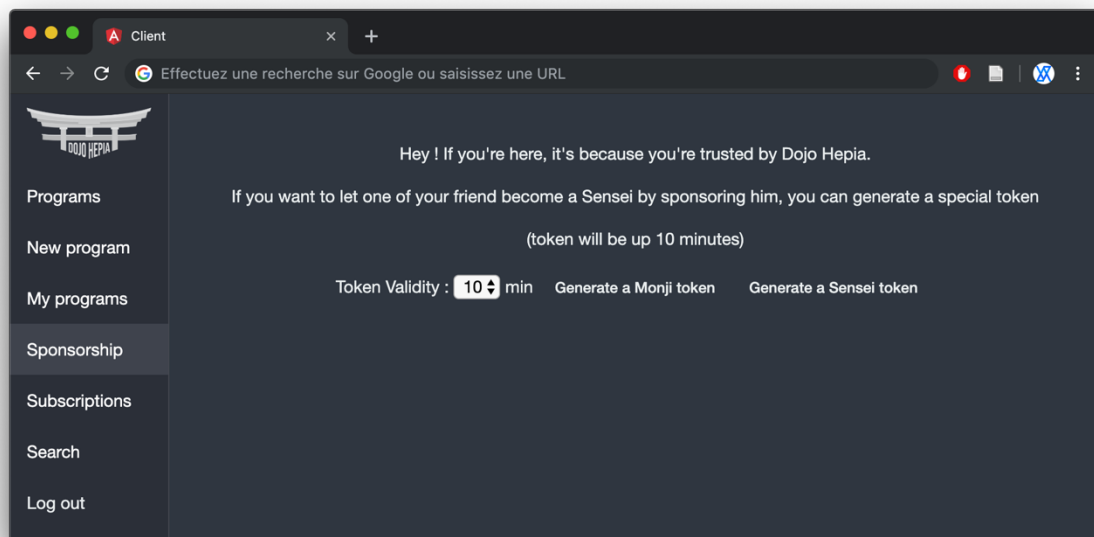


Figure 27 - Manuel utilisateur : Sensei - Page de Sponsoring (AV, 2019)

Une fois que vous aurez cliqué sur « Generate a Monji token » (Générer un token pour un Monji, en français) ou « Generate a Sensei token » (Générer un token pour un Sensei, en français), le token apparaîtra en rouge (voir figure Figure 28), et vous pourrez le copier pour le donner à un utilisateur qui voudrait rejoindre DojoHepia.

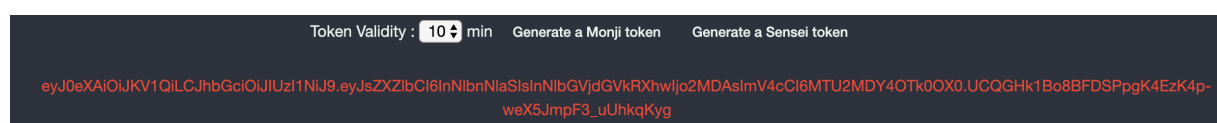


Figure 28 - Manuel utilisateur : Sensei - Token généré (AV, 2019)

En tant que Monji, vous avez la possibilité de :

1. Visualiser les programmes
2. Afficher un programme
3. Vous abonner à un programme
4. Accéder et gérer vos abonnements
5. Réaliser un kata
6. Rechercher des programmes

Visualiser les programmes

En vous authentifiant au site (voir section : n'importe qui – s'authentifier à DojoHepia), vous arrivez sur la page principale de DojoHepia. Celle-ci affiche tous les programmes, le premier étant le plus récent (voir Figure 29).

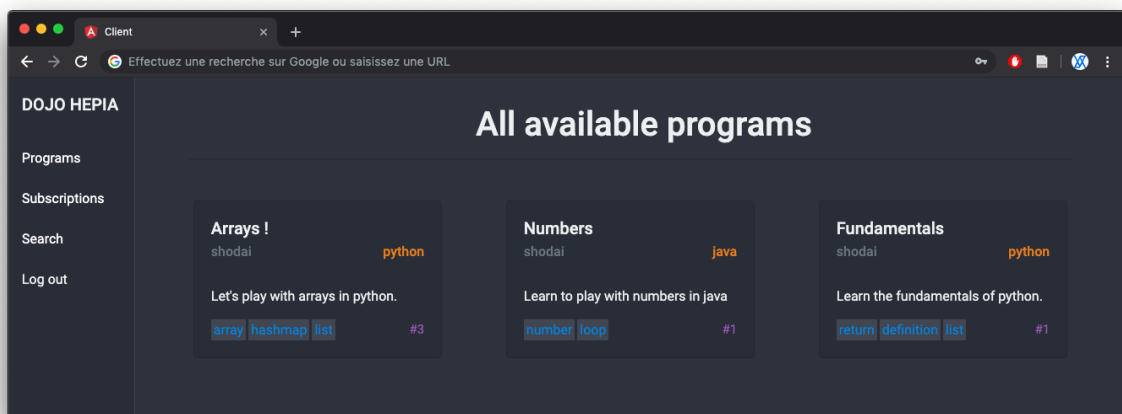


Figure 29 - Manuel utilisateur : Monji - Tous les programmes disponibles (AV, 2019)

Filtrer un programme

Si vous cliquez sur le nom du Sensei, du langage, ou sur un des tags, cela vous permettra de filtrer le programme selon l'entité cliquée (Comme sur les Figures Figure 30 et Figure 31).

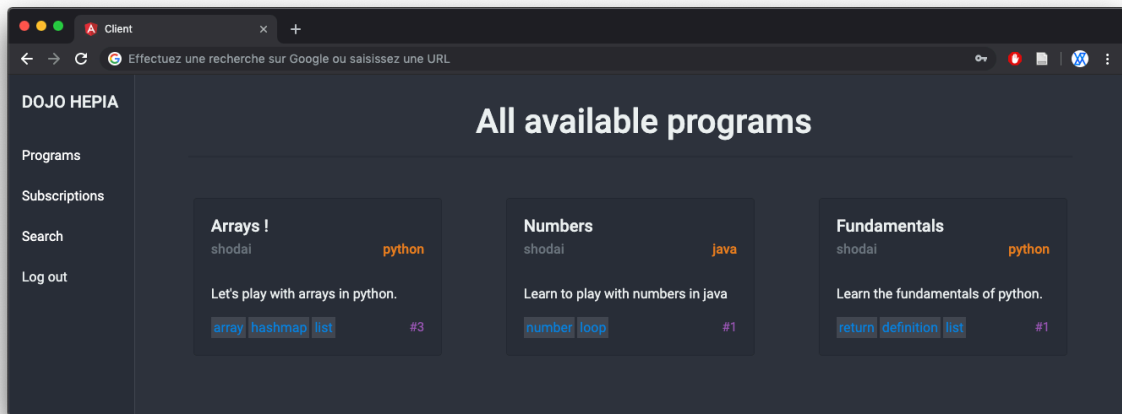


Figure 30 - Manuel utilisateur : Monji - Tous les programmes disponibles | Aucun filtre (AV, 2019)

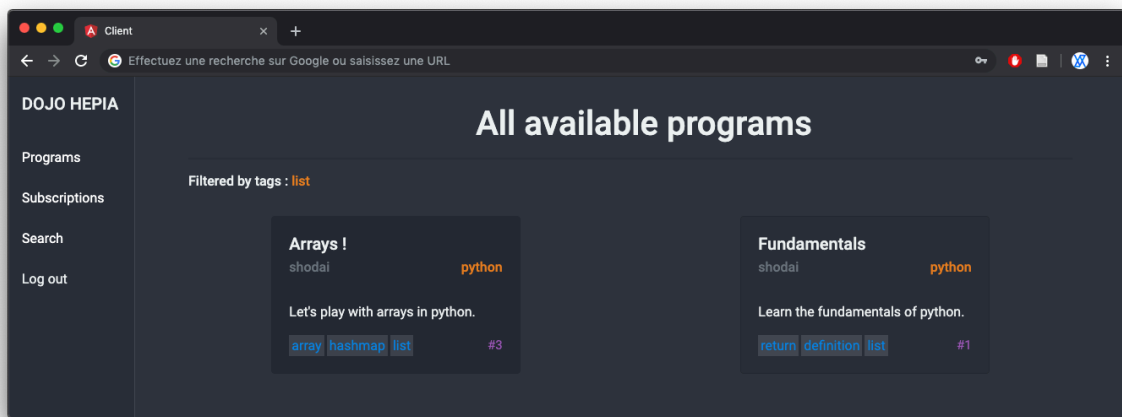


Figure 31 - Manuel utilisateur : Monji -Page : Tous les programmes disponibles | Filtrée par tag (AV, 2019)

Afficher un programme

Lorsque vous cliquez sur un programme, vous accédez à la page de celui-ci.

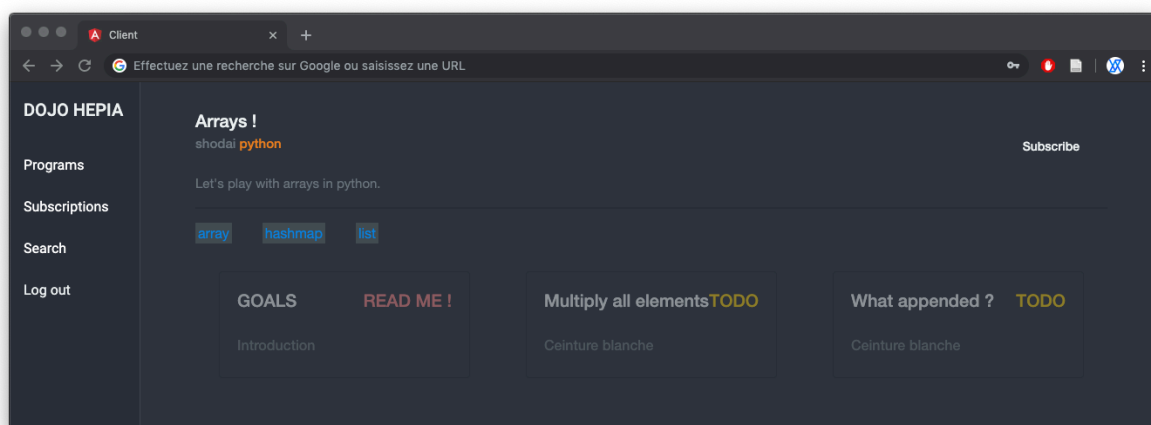


Figure 32 - Manuel utilisateur : Monji - Page d'un kata, utilisateur non-abonné (AV, 2019)

En premier lieu, si vous n'êtes pas abonné au programme les katas vous apparaîtront désactivés (voir Figure 32). Il faudra vous abonner au programme (voir « Vous abonner à un programme ») pour pouvoir les réaliser.

Si vous êtes abonné au programme, il vous apparaîtra comme dans la Figure 33. Dans le bandeau d'entête, les informations affichées (titre, sensei, langage, tags) sont similaires à celles d'une carte programme.

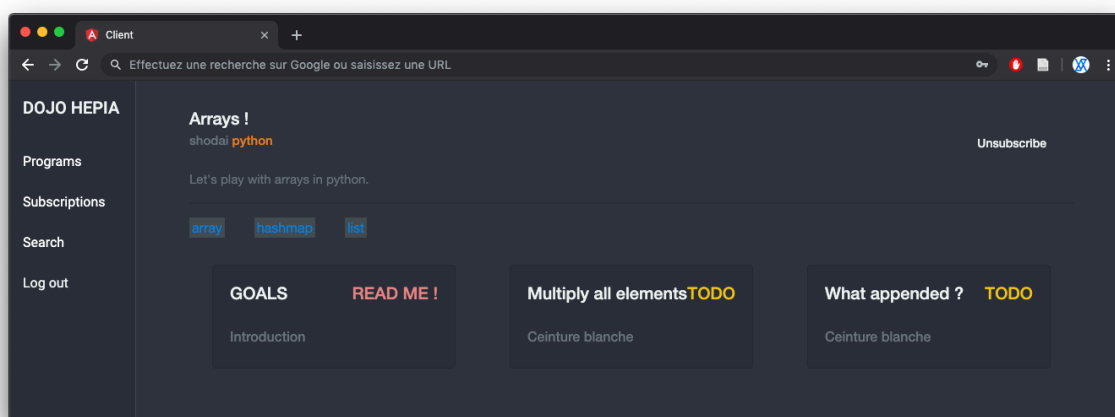


Figure 33 - Manuel utilisateur : Monji - Page d'un kata, utilisateur abonné (AV, 2019)

sur cette vue, vous pourrez voir votre progression sur les différents katas proposés. Les programmes sont composés d'objectifs, ils portent le titre "GOALS", et doivent-être lu avant de réaliser un kata. Chaque katas / objectif à un état de progression, le Tableau 15 étant une liste complète de ceux-ci.

Nom	Action	Description
TODO	S'inscrire à un programme	Vous n'avez pas encore cliqué sur la carte du kata en question
ONGOING	Cliquer sur la carte du kata en question.	Le kata est en cours de réalisation
RESOLVED	Résoudre le kata	Vous avez réussi le kata, votre solution est enregistrée sur le serveur
FAILED	Abandonner le kata	Vous avez abandonné le kata, vous débloquent la solution
READ ME	S'inscrire à un programme	Le kata nommé "GOAL" (optionnel), doit-être lu, c'est les objectifs du programme
DONE	Cliquer sur la carte du GOAL en question.	Vous avez pris connaissance des objectifs du programme
CLOSED	Le Kata est fermé	Le kata est visible, mais n'est plus réalisable

Tableau 15 - Liste des états d'un kata

Vous abonner à un programme

Si vous souhaitez sauvegarder votre progression ou réaliser un kata dans un programme, il vous faut obligatoirement vous abonner à celui-ci. Les programmes auxquels vous êtes abonné ne vous apparaissent pas désactivés et sont visibles sur la page "Mes abonnements".

Mot de passe

Certains programmes nécessitent un mot de passe et vous ne pourrez pas vous abonner au programme sans lui. Le mot de passe vous est généralement fourni par votre Sensei ou un camarade Monji.

Se désabonner

Lorsque vous vous désabonnez d'un programme, il n'est plus visible dans la page "Mes abonnements".

Si vous progressez dans un programme, vos avancements seront sauvegardés même une fois que vous vous désabonnez du programme.

Accéder et gérer vos abonnements

Cette page permet d'afficher vos programmes en cours et vos programmes finis. Un indicateur (%), permet de vous indiquer votre progression sur le programme en question.

Réaliser un kata

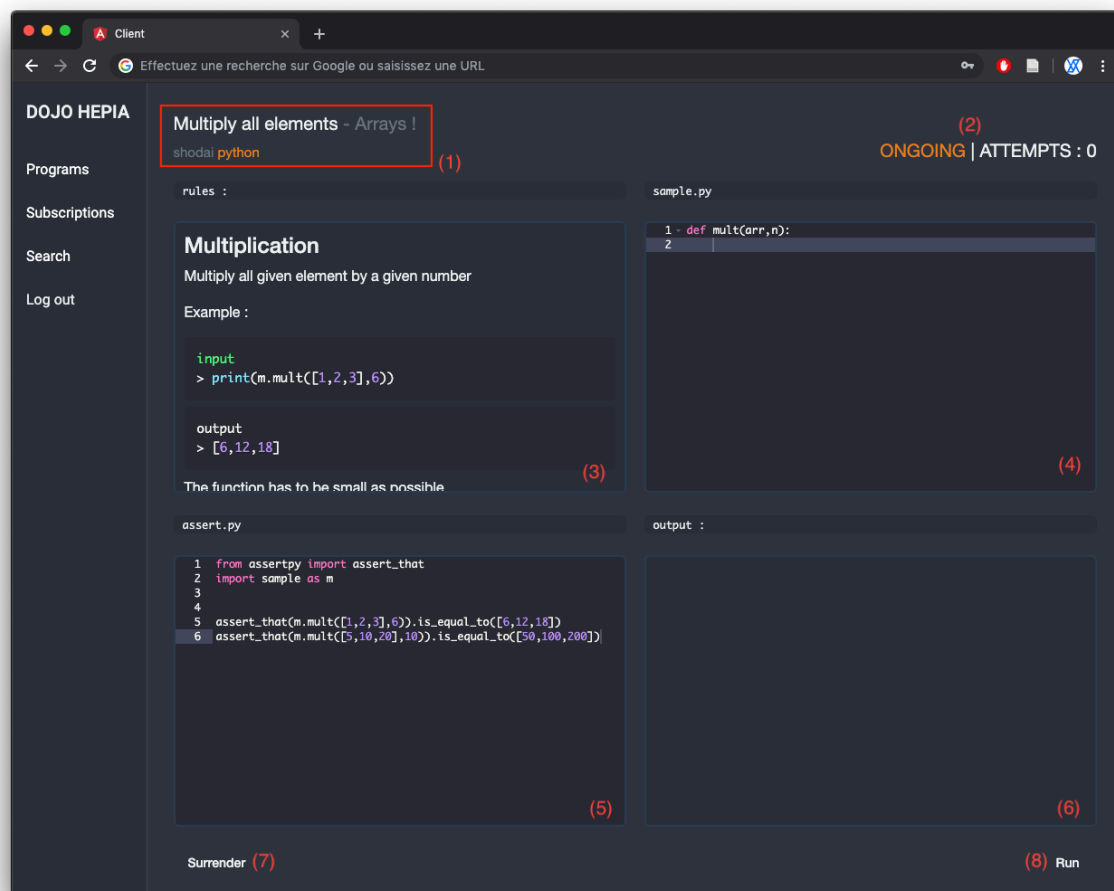


Figure 34 - Manuel utilisateur : Monji - Interface de réalisation d'un kata (AV, 2019)

Voici l'interface de réalisation d'un kata, en voici une description point par point :

1. Le bandeau d'en tête contient le titre du kata, le titre du programme, le sensei et le langage
2. L'état du kata, et le nombre d'essais effectués
3. Les instructions pour la réalisation du kata
4. La boîte ou vous écrivez votre code à valider
5. La batterie de tests, visible ou non
6. Le résultat de l'exécution de votre code avec la batterie de tests
7. Le bouton d'abandon, disponible après le nombre d'essais atteint spécifié par le Sensei
8. Le bouton qui permet de lancer l'exécution du code

A noter qu'une fois que vous avez effectué un kata (abandon ou réussite), vous pourrez le réessayer (un bouton s'affichera à côté du bouton d'abandon (7)), mais le refaire ne vous permettra pas d'en changer l'état.

Rechercher des programmes

La recherche d'un programme s'effectue par le titre de celui-ci.

N'IMPORTE QUI

Rejoindre DojoHepia

Pour créer votre compte utilisateur sur DojoHepia, veuillez cliquer sur le lien “Create my account” (Créer mon compte utilisateur) sur la page de connexion de la plateforme. Vous arriverez alors sur le formulaire suivant :

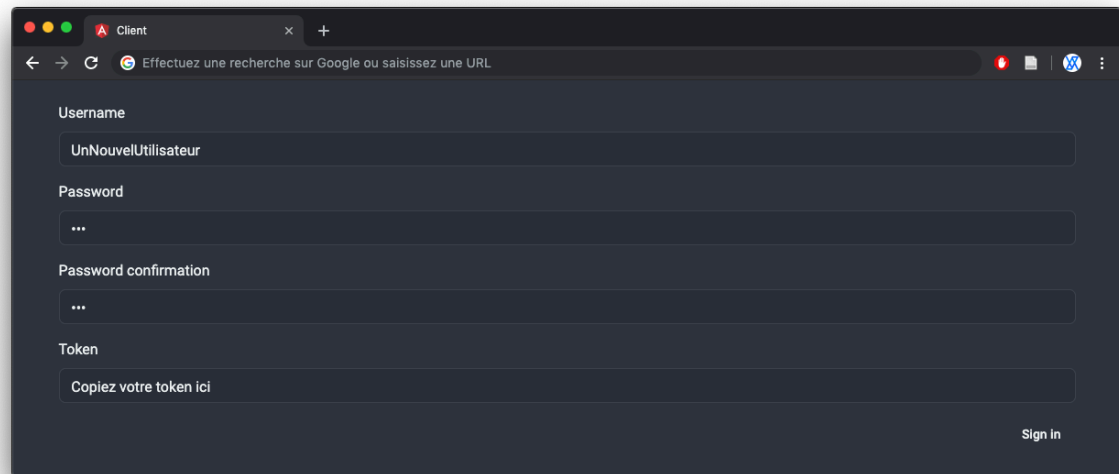
A screenshot of a web browser window showing a dark-themed account creation form. The form has four input fields: 'Username' with the placeholder 'UnNouvelUtilisateur', 'Password' with three dots, 'Password confirmation' with three dots, and 'Token' with the placeholder 'Copiez votre token ici'. A 'Sign In' button is located at the bottom right of the form. The browser's address bar shows a Google search prompt.

Figure 35 - Manuel utilisateur : N'importe qui - Page de création de compte (AV, 2019)

Username (Nom d'utilisateur)

Votre nom d'utilisateur doit-être d'au moins 4 caractères, et ne contenir que les caractères suivants :

AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz 0123456789-

Password & Password Confirmation (Mot de passe et confirmation du mot de passe)

Le mot de passe doit être d'au moins 6 caractères. Vous devez retaper votre mot de passe à l'identique dans “Password confirmation”

Token

Pour rejoindre DojoHepia il faut que l'un des Sensei du site vous ait donné un token d'authentification. Ce token permet de certifier que vous êtes légitime d'accéder à DojoHepia.

Attention : Votre rang d'utilisateur dépend de la nature du token qu'un Sensei vous a donné.

S'authentifier à DojoHepia

Pour vous connecter à DojoHepia, veuillez renseigner les champs “Username” (Nom d'utilisateur) et “Password” (mot de passe), que vous avez spécifié lors de la création de votre utilisateur.

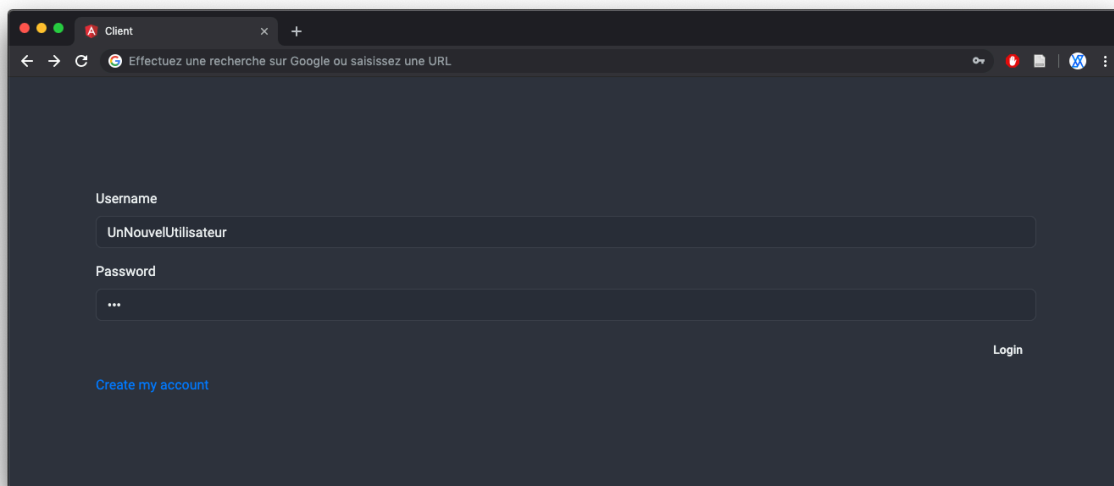


Figure 36 - Manuel utilisateur : N'importe qui - Page d'authentification (AV, 2019)

Votre session de connexion à une validité de 6 jours, après lesquels, il vous faudra vous reconnecter.

ANNEXE 2 : AJOUTER UN LANGAGE POUR LES KATAS

INTRODUCTION

Ajouter un langage de développement dans DojoHepia, n'est pas nécessairement une chose aisée et demande de suivre ce tutoriel à la lettre. Il se divise en 5 parties :

- Image docker
- Mise à jour du Service de compilation Javalin
- Mise à jour du Client Angular
- Tests
- Publication

Image docker

L'exécution des katas est effectuée dans des containers afin de s'affranchir de certains problèmes de sécurité, de ce fait, il vous faut créer un container docker avec les prérequis, pour que votre implémentation fonctionne sur DojoHepia. Voici une courte liste de prérequis :

- Un container simple et léger
- Une librairie d'assertion adaptée à la bonne résolution du kata

Une image simple et légère

Dans le but de créer un container simple et léger, veillez à ne mettre que le strict nécessaire dans votre image. Veillez à n'intégrer que les outils dont vous avez réellement besoin, car chaque petite dépendance aura pour conséquence son lot de ralentissement lors de la création du container par le service de compilation.

La Librairie d'assertion

Nombreux sont les langages proposant déjà des systèmes d'assertion, mais tous ne sont pas forcément adaptés à DojoHepia. En effet, le but de DojoHepia étant d'être pédagogique, il est important que les assertions donnent une description détaillée de l'erreur pour que l'utilisateur

apprenne à les résoudre. Dans le cadre du langage python par exemple, le système d'assertion déjà existant ne fait que d'interrompre le code si le prédicat se révèle faux, mais ne donne en aucun cas plus d'information (d'où l'utilisation d'une librairie externe).

L'image docker

Votre image peut-être basée sur n'importe quelle autre image, mais il faudra en revanche rajouter quelques couches à l'aide d'un Dockerfile. Voici par exemple le Dockerfile de l'image python

```
FROM python
RUN pip3 install assertpy
RUN mkdir /env/
```

Listing 14 - Dockerfile, python

L'image se base sur python. Elle installe une librairie externe pour la batterie tests (assertpy), puis crée un répertoire « env » à la racine de l'image.

Le répertoire partagé

Pour que le container puisse accéder aux fichiers que vous souhaitez faire exécuter, ceux-ci sont stockés dans un répertoire partagé à la racine du service de compilation. C'est pour cela que les Dockerfiles doivent avoir la commande « **RUN mkdir /env/** ».

Il est important d'avoir en tête que le container ne fait pas tout à votre place. Dans le cadre du container python par exemple, il ne suffisait que de ces trois commandes (voir Listing 14), car python n'a pas besoin d'être compilé pour être exécuté. De ce fait, la simple commande « docker run » suivit de « python le_fichier_a_interpreter » suffisait pour que le container exécute le code correctement. Mais dans le cadre du container java, il a fallu d'abord modifier le CLASSPATH, puis compiler les classes ensemble, ainsi qu'avec le JAR de la librairie Junit5, et pour finir lancer l'exécution avec « java Main ». Il est donc de votre devoir de faire les tests nécessaires pour que le container s'exécute de la manière souhaitée. Une chose importante toutefois, c'est que votre flow d'exécution générale ne contienne que deux fichiers (voir plus bas).

Mise à jour du Service de compilation Javalin

Les modifications se portent exclusivement sur le fichier « DockerCompilation.java » dans le service de compilation (*dojo-hepia/compilation/src/main/java/DockerCompilation.java*).

Modifier le dictionnaire des langages

Il vous faut modifier le dictionnaire des langages qui se trouve dans le constructeur de la classe Docker compilation, comme dans le Listing 15.

```
public DockerCompilation() {
    // nl = nouveau langage
    Map<String, String> nl = new HashMap<>();
    nl.put("filename", "share_docker_file/fichier_code.ext");
    nl.put("test", "share_docker_file/fichier_batterie_tests.ext");
    nl.put("cmd", "docker run --rm --mount type=bind,source=" +
        AbsolutePathToFile + "/share_docker_file,dst=/env/ IMAGE_NOUVEAU_LANGAGE
        [...] ");
    [...]
    this.filenameDic.put("nl", nl);
}
```

Listing 15 - Exemple d'ajout d'une nouvelle entrée dans le dictionnaire des langages

« fichier_code.ext » : C'est le fichier où est écrit le code tapé par l'utilisateur.

« fichier_batterie_tests.ext » : C'est le fichier où est écrit la batterie de tests

« IMAGE_NOUVEAU_LANGAGE [...] » : C'est le tag de votre image, plus les options de lancement adéquates.

Les options de lancement adéquates

« fichier_code.ext » & « fichier_batterie_tests.ext » sont les fichiers qui seront écrits par le service de compilation dans le répertoire partagé, ça n'est donc pas à vous de vous en occuper. En revanche, vous devez veiller à ce qu'ils soient utilisés de manière cohérente avec l'image que vous avez créée. Par exemple pour python, il y a le fichier « sample.py » (code écrit), et « assert.py » (batterie de tests), « assert.py » est le fichier principal, c'est dans celui-ci que sont importées (au besoin) les dépendances nécessaires à l'exécution. Comme « from sample import * » (voir Listing 8), qui est le code écrit par l'utilisateur (son importation en python est primordiale pour pouvoir utiliser ses fonctions).

Mise à jour du Client Angular

Une partie plus simple, mais pas moins importante, est de mettre à jour l'interface du Client et certains fichiers concernés.

La modification se porte sur les fichiers suivants :

- dojo-hepia/client/src/app/languages_canvas.ts
- dojo-hepia/client/src/app/component/program-create/program-create.component.html

languages_canvas.ts

Ce fichier (voir Listing 16) contient les informations nécessaires pour la création d'un kata, à savoir :

- id : L'id du langage (nom du langage, exemple : 'python', 'java')
- assertCanva : Le canevas de batterie de tests, qui est le bout de code donné au Sensei pour qu'il crée sa batterie de tests (avec l'importation des librairies concernées aussi)
- codeCanva : Le canevas dans lequel le Sensei va écrire sa solution
- filename : Nom du fichier où l'utilisateur va écrire sa solution (Seulement à titre d'affichage, visible sur l'affichage d'un kata)
- assertname : Nom du fichier où le Sensei va écrire sa batterie de tests (Seulement à titre d'affichage, visible sur l'affichage d'un kata)

```
export const LANG: Canva[] = [
  {
    id: 'python',
    assertCanva: 'from assertpy import assert_that\nfrom sample import *' +
    '\n\n' +
    '# Example : \n#
    assert_that(yourfunction(someValues)).is_equal_to(targetedValues)',
    codeCanva: '# Write your code here',
    filename: 'sample.py',
    assertname: 'assert.py'
  },
  {...}
];
```

Listing 16 - Entrée d'un langage dans le fichier LANG

Il vous suffit alors d'ajouter une nouvelle entrée (à la place de {...}) en remplissant tous les champs nécessaires.

Note : Veillez à écrire les importations des fichiers comme dans « assertCanva ».

[program-create.component.html](#)

Ce fichier est le front-end de la page de création d'un programme, ici, il vous faut seulement rajouter une « option » dans la balise « select » qui concerne les langages. La valeur de l'option doit correspondre à l'id du langage décrit dans le fichier [languages_canva.ts](#).

Exemple :

```
<label for="language">Targeted language</label>
  <select class="form-control" FormControlName="language"
    [ngClass]="{ 'is-invalid': submitted && f.language.errors}">
    <option value="python">python</option>
    <option value="java">java</option>
  </select>
```

Listing 17 - Ajout d'un kata, le code, avant l'ajout d'un langage

Devient alors :

```
<label for="language">Targeted language</label>
  <select class="form-control" FormControlName="language"
    [ngClass]="{ 'is-invalid': submitted && f.language.errors}">
    <option value="python">python</option>
    <option value="java">java</option>
    <option value="nl">nl</option> <-----
  </select>
```

Listing 18 - Ajout d'un kata, le code, après l'ajout d'un langage

Tests

Dans le cadre d'une bonne expérience utilisateur, veillez à tester sur DojoHepia votre nouvelle implémentation du langage avant de la publier. De bons tests à effectuer seraient les suivants :

- Créer un kata : hello world
- Une erreur de syntaxe dans le code
- La fonction ne remplit pas le prédicat écrit par le Sensei
- Que les temps d'attentes entre l'appui du bouton « run » et le résultat ne soient pas excessifs
- Que l'ajout de librairie interne au langage ne pose pas de soucis

Publication

Si votre image est prête et que tous les tests sont au vert, vous pouvez désormais publier votre nouveau langage sur la plateforme en veillant toutefois d'être à l'écoute des retours utilisateurs lors des premières semaines qui suivent la publication du langage.

ANNEXE 3 : LES SERVICES ANGULAR

Cette annexe permet de décrire les services Angular.

On peut catégoriser les services créés pour DojoHepia en 5 parties :

- Les programmes
 - Interactions avec un programme
 - Les abonnements aux programmes
 - La recherche de programmes
- Les katas
 - Interactions avec un kata
 - Les abonnements aux katas
- La compilation
- L'authentification
 - Connection, inscription, génération de token
- Le service LANG
 - Récupération du code canevas lors de la création d'un kata

Pour une question de lisibilité, voici un tableau qui décrit les abréviations utilisées dans la description des services :

Acronyme	Signification
pid	Identifiant de l'utilisateur
pid	Identifiant du programme
obj	Contenu utile à certaines interactions
p	Un programme
pwd	Mot de passe
k	Un kata
kid	L'identifiant d'un kata
did	L'identifiant d'un document
usrn	Username – Nom de l'utilisateur

LES PROGRAMMES

Nom du service (.service.ts)	Méthodes	Description
Interactions avec un programme		
program	create(obj)	Crée un programme
	get()	Récupère tous les programmes
	getById(pid)	Récupère le programme correspondant à un ID.
	delete(pid)	Supprime un programme
	isOwner(pid,uid)	Prédicat, l'utilisateur est-il le créateur du programme ?
	update(pid, p)	Met à jour un programme
	duplicate(pid, nid, title)	Duplique un programme
	check(pid, pwd)	Vérifie si le mot de passe fourni par l'utilisateur correspond bien au mot de passe du programme.
Les abonnements aux programmes		
programme-subscription	create(uid, obj)	Crée l'abonnement d'un utilisateur à un programme
	getMine(uid)	Récupère tous les programme créés par un Sensei
	getSubscription(uid)	Récupère les abonnements d'un utilisateur
	getSubs(pid,uid)	Récupère la progression d'un abonné sur un programme
	toggle(obj)	Change le statut d'abonnement d'un utilisateur à un programme
La recherche de programmes		
fetch-program-by-type	getPrograms(type, res)	Récupère les programmes dont le contenu du champ « type » correspond à « res »

LES KATAS

Nom du service (.service.ts)	Méthodes	Description
Interactions avec un kata		
kata	publish(k,pid,isGoal)	Crée un kata
	getKatasDetails(pid,uid)	Récupère les katas d'un programme, avec la progression d'un utilisateur
	get(kid,pid)	Récupère le kata
	delete(kid,pid)	Supprime un kata
	toggleActivation(kid,pid)	Change le statut d'activation d'un kata
	isOwner(kid,uid,pid)	Prédicat, l'utilisateur est-il le créateur du kata ?
	update(obj,pid)	Met à jour un kata
	isActivated(kid,pid)	Donne l'état du statut de l'activation d'un kata
	upload(file)	Permet d'envoyer un document vers le serveur
	getDocument(did,pid)	Récupère le document d'un kata
Les abonnements aux katas		
kata-subscription	create(obj)	Crée la première sauvegarde d'un utilisateur sur un kata
	increment(obj)	Augmente le nombre d'essais de 1, d'un utilisateur sur un kata
	get(kid,pid,uid)	Récupère l'état de la progression d'un utilisateur sur un kata
	update(obj)	Met à jour la progression d'un utilisateur sur un kata
	isSubscribed(uid,pid)	Récupère l'état de l'abonnement d'un utilisateur

LA COMPILATION

Nom du service (.service.ts)	Méthodes	Description
Compilation du code		
compilation	compilationServer(obj)	Demande au Gateway d'exécuter du code

L'AUTHENTIFICATION

Nom du service (.service.ts)	Méthodes	Description
Connection		
authentification	login(usrn,pwd)	Authentification
	logout()	Destruction du localStorage
Inscription		
sign-in	createUser(obj)	Crée un utilisateur
Génération de token		
token	getToken(level,time)	Demande au Gateway de générer un token pour la création d'un compte

LANG

Nom du service (.service.ts)	Méthodes	Description
Lang		
lang	getLang(id)	Récupère le canevas de l'id concerné

ANNEXE 4 : VERSIONNING

Le versionning est un suivi de l'évolution de l'application.
Tout au long du développement, DojoHepia était maintenue sur GitHepia.

Note : Ne sont montrés que les points importants du développement.

Nom de version	Date de publication	Ajout
DojoHepia-v0.1	8 Mai 2019	<p>Serveur de compilation</p> <ul style="list-style-type: none">- Containerisation- Support Java- Support python <p>Client Angular</p> <ul style="list-style-type: none">- Ajout d'un loader- Ajout d'une page 404- Ajout de programme- Ajout de kata- Ajout du plugin <i>Ace editor</i>- Affichage des programmes- Affichage des katas <p>Gateway</p> <ul style="list-style-type: none">- In memory database
DojoHepia-v0.2	9 Mai 2019	Il est désormais possible de lancer les serveurs Javalin avec <i>mvn</i>
DojoHepia-v0.3	15 Mai 2019	<p>Gateway</p> <ul style="list-style-type: none">- Plus d'in memory database- MongoDB- Authentification avec JWT- A une notion d'utilisateurs <p>Client Angular</p> <ul style="list-style-type: none">- Filtrage des programmes- Authentification avec JWT- A une notion d'utilisateurs- Les pages ne sont accessibles qu'avec le bon rang- Abonnement aux programmes- Page : Mes abonnements- Page : Mes programmes- Terminer un kata sauvegarde sa solution dans le serveur- Option : Abandon

		<ul style="list-style-type: none"> - De nouveaux états de progression sont visibles dans l’affichage des katas - L’utilisateur peut visualiser le nombre de katas qu’il a effectué - Recherche des programmes par leur titre
DojoHepia-v0.4	16 Mai 2019	Gateway <ul style="list-style-type: none"> - Création d’utilisateurs - Les tokens sont valables 6 jours Client Angular <ul style="list-style-type: none"> - Création d’utilisateur - Un Sensei peut générer un token pour la création d’un Sensei MongoDB <ul style="list-style-type: none"> - Ajout d’un niveau « Users »
DojoHepia-v0.4.5	16 Mai 2019	Ajout du README, pour GitHepia
DojoHepia-v0.5	29 Mai 2019	Client Angular <ul style="list-style-type: none"> - Une bottom sheet s’affiche pour les options d’un kata - Suppression de programmes - Suppression de katas - Modification de katas - Support du markdown ajouté pour la rédaction d’instructions - Désactivation de katas - L’utilisateur peut visualiser ses programmes finis ou en cours - Duplication de programmes
DojoHepia-v1.0	8 juillet 2019	Gateway <ul style="list-style-type: none"> - Implémentation de Future Client Angular <ul style="list-style-type: none"> - Kata objectif - Fermeture de katas - Upload de documents à la place de la rédaction de règles - Mot de passe sur les programmes - La bottom sheet n’existe plus, maintenant c’est un petit menu déroulant (kata) - Menu déroulant pour la gestion de programmes Service de compilation <ul style="list-style-type: none"> - Mise à jour de l’image python (Docker)

ANNEXE 5 : MODULES UTILISÉS PAR DOJOHEPIA

Nom	Description	Licence
Angular	Framework web	MIT
assertpy	Librairie d'assertion pour python	BSD
Bootstrap	Librairie d'interface utilisateur	MIT
Brace	Librairie contenant des codes couleurs pour ace editor	MIT
Javalin	Framework web pour créer un serveur HTTP	Apache
js-sha1	Permet de convertir du texte en sha1	MIT
Junit	Librairie d'assertion pour java	EPL
ng2-ace-editor	Éditeur de code live intégrer au web	MIT
ngx-alerts	Gestionnaire d'alertes side-pop-up	MIT
ngx-bootstrap	Librairie d'interface utilisateur pour Angular	MIT
ngxmarkdown	Librairie permettant le support de markdown pour les katas	MIT
uuid	Librairie générant des ID uniques	MIT

ANNEXE 6 : UTILISER DOJOHEPIA

Cette annexe est un tutoriel pour l'installation et l'utilisation de DojoHepia. Le projet est accessible à l'adresse suivante :

<https://githopia.hesge.ch/alexandr.vanini/dojo-hepia>

Pour pouvoir continuer ce tutoriel, il vous faut les prérequis suivants :

- MAVEN
- DOCKER
- DOCKER-COMPOSE
- ANGULAR CLI

1. LA BASE DE DONNÉES

Répertoire : ./mongodb/

```
docker-compose up -d
```

Listing 19 : Utiliser DojoHepia - Commande pour lancer la base de données

Note : La base de données est pré-popularisée

2. LE GATEWAY

Répertoire : ./gateway/

```
mvn package  
mvn exec:java
```

Listing 20 - Utiliser DojoHepia - Compiler et lancer le Gateway

3. LE CLIENT

Répertoire : ./client/

```
npm install
ng serve --open
```

Listing 21 - Utiliser DojoHepia - Compiler et lancer le Client

4 LE SERVICE DE COMPILATION

4.1 Téléchargement de l'image pour le container java

```
docker pull freakency/java:1.0
```

Listing 22 - Utiliser DojoHepia - Télécharger l'image pour le container java

4.2 Téléchargement de l'image pour le container python

```
docker pull freakency/python:3.0
```

Listing 23 - Utiliser DojoHepia - Télécharger l'image pour le container python

4.3 Execution

Répertoire : ./compilation/

```
mvn package
mvn exec:java
```

Listing 24 - Utiliser DojoHepia - Compiler et lancer le service de compilation

5 LISTE DES UTILISATEURS DÉJÀ PRÉSENT

DojoHepia contient déjà des données lors de son premier lancement (si vous avez correctement suivi le tutoriel d'installation), dont des utilisateurs :

Nom de l'utilisateur	Mot de passe	Privilège
Shodai	Admin	Shodai
Noob	Noob	monji

Tableau 16 - Utiliser DojoHepia - Utilisateurs déjà présents

RÉFÉRENCES DOCUMENTAIRES

Ace - Code Editor for the Web, 2019. [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <https://ace.c9.io/>

Angular - Introduction to components, 2019. [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <https://angular.io/guide/architecture-components>

Angular - Observables, 2019. [en ligne]. [Consulté le 28 mai 2019]. Disponible à l'adresse : <https://angular.io/guide/observables>

Angular - Services, 2019. [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <https://angular.io/tutorial/toh-pt4>

Angular, 2019. [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <https://angular.io/>

Assertion, 2018. *Wikipédia* [en ligne]. [Consulté le 4 juin 2019]. Disponible à l'adresse : <https://fr.wikipedia.org/w/index.php?title=Assertion&oldid=149971675>
Page Version ID: 149971675

Codewars, 2019. *Codewars* [en ligne]. [Consulté le 16 avril 2019]. Disponible à l'adresse : <http://www.codewars.com>
Codewars is where developers achieve code mastery through challenge. Train on kata in the dojo and reach your highest potential.

CodinGame, 2019. *CodinGame* [en ligne]. [Consulté le 2 avril 2019]. Disponible à l'adresse : <https://www.codingame.com/>
CodinGame est une plate-forme de jeux et concours pour les programmeurs, où vous pouvez améliorer vos compétences avec des exercices intéressants (+ de 25 langages supportés)

DIALLO, Daouda, 2018. Création d'une Application Web avec NodeJS et Angular V6 — Partie 2 (FrontEnd). *Medium* [en ligne]. 6 juillet 2018. [Consulté le 17 avril 2019]. Disponible à l'adresse : <https://medium.com/code-divoire/creation-dune-application-web-avec-node-js-et-angular-v6-partie-2-frontend-820c998f6996>

Docker documentation - Dockerfile, 2019. *Docker Documentation* [en ligne]. [Consulté le 17 juin 2019]. Disponible à l'adresse : <https://docs.docker.com/engine/reference/builder/>
Dockerfiles use a simple DSL which allows you to automate the steps you would normally manually take to create an image.

Docker Hub, 2019. [en ligne]. [Consulté le 17 juin 2019]. Disponible à l'adresse : <https://hub.docker.com/>

Docker run - Documentation, 2019. *Docker Documentation* [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <https://docs.docker.com/engine/reference/run/>
Configure containers at runtime

Docker, 2019. *Docker* [en ligne]. [Consulté le 16 juin 2019]. Disponible à l'adresse : <https://www.docker.com/>

Build, Share, and Run Any App, Anywhere. Learn about the only enterprise-ready container platform to cost-effectively build and manage your application portfolio.

HACHET, Nicolas, 2012. L'architecture REST expliquée en 5 règles. *Blog PHP de Nicolas Hachet* [en ligne]. 15 juin 2012. [Consulté le 27 juin 2019]. Disponible à l'adresse : <https://blog.nicolashachet.com/niveaux/confirmes/larchitecture-rest-expliquee-en-5-regles/>

How to convert a file to a blob in java / How to convert a file into a byte array in java | coDippa, 2019. [en ligne]. [Consulté le 6 juin 2019]. Disponible à l'adresse : <https://codippa.com/how-to-convert-a-file-to-byte-array-in-java/>

How to Preview Blobs With HTTP POST and Angular 5 - DZone Web Dev, 2019. *dzone.com* [en ligne]. [Consulté le 6 juin 2019]. Disponible à l'adresse : <https://dzone.com/articles/how-to-preview-blobs-with-http-post-and-angular-5>

A software architect gives a brief tutorial on how to use Angular to create an HTTP POST call that will upload an image (or blob) to a basic HTML web page.

Java SE Development Kit 11, 2019. [en ligne]. [Consulté le 17 juin 2019]. Disponible à l'adresse : <https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html>

Javalin, 2019. *Javalin* [en ligne]. [Consulté le 17 avril 2019]. Disponible à l'adresse : <https://javalin.io>

Javalin - A lightweight Java and Kotlin web framework. Create REST APIs in Java or Kotlin easily.

JDoodle - free Online Compiler, Editor for Java, C/C++, etc, 2019. [en ligne]. [Consulté le 16 juin 2019]. Disponible à l'adresse : <https://www.jdoodle.com>

JSON Web Token, 2018. *Wikipédia* [en ligne]. [Consulté le 14 mai 2019]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=JSON_Web_Token&oldid=152446495
Page Version ID: 152446495

JUnit 5, 2019. [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <https://junit.org/junit5/>

JWT.IO - JSON Web Tokens Introduction, 2019. [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <http://jwt.io/>

Learn about JSON Web Tokens, what are they, how they work, when and why you should use them.

Lexique des arts martiaux, 2019. *Saintes Karaté Club* [en ligne]. [Consulté le 28 mai 2019]. Disponible à l'adresse : <http://www.sainteskarateclub.com/non-categorise/lexique-des-arts-martiaux.html>

MongoDB Manual, 2019. <https://github.com/mongodb/docs/blob/v4.0/source/reference/sql-comparison.txt> [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <https://docs.mongodb.com/manual/reference/sql-comparison>

MongoDB, 2019. [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <https://www.mongodb.com/fr>

ng2-ace-editor, 2019. *npm* [en ligne]. [Consulté le 2 mai 2019]. Disponible à l'adresse : <https://www.npmjs.com/package/ng2-ace-editor>
Ace editor integration with typescript for Angular.

Open Source Document Database, 2019. *MongoDB* [en ligne]. [Consulté le 2 juillet 2019]. Disponible à l'adresse : <https://www.mongodb.comnull>

PRITCHARD, Adam, 2019. *Markdown cheat sheet* [en ligne]. JavaScript. [Consulté le 27 juin 2019]. Disponible à l'adresse : <https://github.com/adam-p/markdown-here>

Scalability, 2018. *Wikipédia* [en ligne]. [Consulté le 15 juin 2019]. Disponible à l'adresse : <https://fr.wikipedia.org/w/index.php?title=Scalability&oldid=154430879>
Page Version ID: 154430879

SHACKLETTE, Justin, 2019. *assertpy: Assertion library for python unit testing with a fluent API* [en ligne]. Python. [Consulté le 4 juin 2019]. Disponible à l'adresse : <https://github.com/ActivisionGameScience/assertpy>

Use volumes, 2019. *Docker Documentation* [en ligne]. [Consulté le 16 juin 2019]. Disponible à l'adresse : <https://docs.docker.com/storage/volumes/>

Using JWT with a Javalin application - Javalin - A lightweight Java and Kotlin web framework. Create REST APIs in Java or Kotlin easily., 2019. *Javalin* [en ligne]. [Consulté le 2 mai 2019]. Disponible à l'adresse : <https://javalin.io>
This is a simple tutorial on how to integrate JWT into a Javalin application. It relies on an extension which can be found here.

What is HTML5 Local Storage? - Definition from Techopedia, 2019. *Techopedia.com* [en ligne]. [Consulté le 28 mai 2019]. Disponible à l'adresse : <https://www.techopedia.com/definition/27674/html5-local-storage>

HTML5 Local Storage Definition - HTML5 local storage is a component of the Web storage app

What is Scrum?, 2019. [en ligne]. [Consulté le 25 juin 2019]. Disponible à l'adresse : https://www.scrum.org/resources/what-is-scrum?gclid=Cj0KCQjwjMfoBRDDARIsAMUjNZrZpM7KCVWdkOyvlabOJIAJW-DgA3nSEBDmSBRqV3npwMOvIvjoo3waAIT7EALw_wcB