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-e HES responsable  **Joël Cavat** | Mandant (si existant)  **HEPIA** |

Sommaire détaillé

Sommaire détaillé 2

Remerciements 6

Énoncé du sujet 7

Résumé 8

Table des illustrations 9

Table des listing 9

Index des tableaux 10

Liste des acronymes 11

Introduction 12

Avant-propos 12

Chapitre 1 : Analyse fonctionnel 16

Analyse du marché et des besoins 16

Codewars 16

CodinGame 17

Points relevés 18

Game-ification 19

Chapitre 2 : Conception 20

Préambule 20

Architecture 20

Client 21

Gateway 21

Compilation 21

Data Provider 21

Vision à long terme 21

Choix technologiques 23

Angular 23

Javalin 23

MongoDB 23

Base de données 24

Users 24

Programs 26

Chapitre 3 : Mise en œuvre 27

Préambule 27

Le Gateway 27

Services Angular 28

Les programmes 28

Les katas 29

Les Programmes 31

Implémentation 32

Filtrer un programme 32

Recherche 33

Gestion 33

Les katas 33

Les états 34

Interface de réalisation d’un Kata 35

Exécution du code 38

Le service de compilation 38

Implémentation 41

Communication entre les différents services 45

Réussir un kata 46

Abandonner un kata 46

Gestion 47

Gestion des ressources et privilèges 48

JWT 48

Implémentation 50

Les abonnements 55

Conclusion 57

ANNEXES 58

Annexes 1 : Tests et Résultats 59

Annexe 2 : manuel de l’utilisateur 60

Sensei 60

Créer un programme 60

Gérer votre programme 60

Créer un kata 60

Gérer votre kata 60

Visualiser vos programmes 60

Sponsoriser un nouvel utilisateur 61

Monji 61

Visualiser les programmes 62

Afficher un programme 63

Vous abonner à un programme 65

Accéder et gérer vos abonnements 66

Réaliser un kata 66

Rechercher des programmes 67

N’importe qui 67

Rejoindre DojoHepia 67

S’authentifier à DojoHepia 68

Annexe 3 : Ajouter un langage pour les katas 69

Introduction 69

Image docker 69

Mise à jour du Service de compilation Javalin 71

Mise à jour du client Angular 72

Tests 73

Publication 73

Annexe 4 : Utiliser DojoHepia 74

Références documentaires 75

*Memento mori.*

Remerciements

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 Delya, qui aura rendu ces trois mois plus facile.

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

Énoncé du sujet

HEPIA «Dojo» a pour but de réaliser un outil à but pédagogique pour l’apprentissage de langages de programmation. Il se base sur des idées existantes, non libre, 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 qualité́ du code, la modularité́, sa réutilisation et sa maintenance. Le principe de « Gamification » sera utilisé (étudié ?) 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 échoué

|  |  |
| --- | --- |
| Candidat :  **VANINI Alexandre**  Filière d’études : ITI | Professeurs responsables :  **CAVAT Joël**  **MALASPINAS Orestis**  En collaboration avec : -  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 aujourd’hui, et une multitude d’outils voient le jour. Ces mêmes outils se sont rendus indispensables et sont maintenant omniprésents dans nos vies et dans notre société. En effet, du simple beeper à la Blockchain, ils trouvent leurs applications partout. Cette croissance a permis de naturellement , mais non sans peine, à orienter l’apprentissage scolaire vers une migration informatique de masse, permettant aux élèves d’accéder à une multitude d’outils et de techniques permettant de graduellement améliorer leur apprentissage des compétences et connaissances.

Hepia n'a pas dérogé à cette croissance et se voit doté 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’informations, 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 aux élèves de réaliser des exercices, ne nécessitant pas d’installer des outils ou Frameworks pour leur réalisation, mais bien d’avoir son propre outil de programmation et 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 intention toute particulière est portée sur l’étude des technologies de développement Web les mieux cotées du moment, ainsi que sur le design et l’expérience utilisateur, pour permettre à DojoHepia de devenir et rester une plateforme fiable et simple d’utilisation.

Table des illustrations

[Figure 1 - Architecture de DojoHepia 21](#_Toc12274093)

[Figure 2 - La collection Users et ses sous-collections (AV, 2019) 25](#_Toc12274094)

[Figure 3 - La collection Programs et sa sous-collection (AV, 2019) 27](#_Toc12274095)

[Figure 4 - Vue des programmes (AV, 2019) 32](#_Toc12274096)

[Figure 5 - Carte d'un programme (AV, 2019) 32](#_Toc12274097)

[Figure 6 - Vue des programmes, aucun filtre (AV,, 2019) 33](#_Toc12274098)

[Figure 7 - Vue des programmes, Filtre sur tag "list" appliqué (AV, 2019) 33](#_Toc12274099)

[Figure 9 - Vue des katas, Monji (AV, 2019) 34](#_Toc12274100)

[Figure 10 - Uu kata, utilisateur Sensei (AV, 2019) 35](#_Toc12274101)

[Figure 11 - Vue de la réalisation d'un kata (AV, 2019) 37](#_Toc12274102)

[Figure 12 – Réalisation d’un kata, communication entre les services (AV, 2019) 46](#_Toc12274103)

[Figure 13 - Gestion d'un kata, menu déroulant 48](#_Toc12274104)

[Figure 14 - Authentification d’un client au serveur (AV, 2019) 52](#_Toc12274105)

[Figure 15 - Cycle de vie d’une requête http (AV, 2019) 53](#_Toc12274106)

[Figure 16 - Manuel utilisateur : Sensei - Page de Sponsoring (AV, 2019) 62](#_Toc12274107)

[Figure 17 - Manuel utilisateur : Sensei - Token généré (AV, 2019) 62](#_Toc12274108)

[Figure 18 - Manuel utilisateur : Monji - Tous les programmes disponibles (AV, 2019) 63](#_Toc12274109)

[Figure 19 - Manuel utilisateur : Monji - Tous les programmes disponibles | Aucun filtre (AV, 2019) 64](#_Toc12274110)

[Figure 20 - Manuel utilisateur : Monji -Page : Tous les programmes disponibles | Filtrée par tag (AV, 2019) 64](#_Toc12274111)

[Figure 21 - Manuel utilisateur : Monji - Page d'un kata, utilisateur non-abonné (AV, 2019) 65](#_Toc12274112)

[Figure 22 - Manuel utilisateur : Monji - Page d'un kata, utilisateur abonné (AV, 2019) 65](#_Toc12274113)

[Figure 23 - Manuel utilisateur : Monji - Interface de réalisation d'un kata 67](#_Toc12274114)

[Figure 24 - Manuel utilisateur : N'importe qui - Page de création de compte (AV, 2019) 68](#_Toc12274115)

[Figure 25 - Manuel utilisateur : N'importe qui - Page d'authentification 69](#_Toc12274116)

Table des listing

[Listing 1 - Dockerfile de l'image python 43](#_Toc11943151)

[Listing 2 - Dockerfile de l'image java 44](#_Toc11943152)

[Listing 3 - Contenu du fichier .bashrc 44](#_Toc11943153)

[Listing 4 - Contenu du fichier java\_test.sh 44](#_Toc11943154)

[Listing 5 - Commande docker, permettant de créer et lancer un container python 45](#_Toc11943155)

[Listing 6 - Commande docker, permettant de créer et lancer un container java 45](#_Toc11943156)

[Listing 7 - Un JWT 49](#_Toc11943157)

[Listing 8 - Header d’un JWT 50](#_Toc11943158)

[Listing 9 - Exemple du payload d’un JWT 51](#_Toc11943159)

[Listing 10 - Dockerfile, python 71](#_Toc11943160)

[Listing 11 - Exemple d'ajout d'une nouvelle entrée dans le dictionnaire des langages 72](#_Toc11943161)

[Listing 12 - Entrée d'un langage dans le fichier LANG 73](#_Toc11943162)

[Listing 13 - Ajout d'un kata, le code, avant l'ajout d'un langage 74](#_Toc11943163)

[Listing 14 - Ajout d'un kata, le code, après l'ajout d'un langage 74](#_Toc11943164)

Index des tableaux

[Tableau 1 - Les programmes par notion / par langage 32](#_Toc12277268)

[Tableau 2 - Les differents status d'un kata 36](#_Toc12277269)

[Tableau 3 - Exemple d'utilisation d'assert py 40](#_Toc12277270)

[Tableau 4 - Résultat de l'exécution d'un kata, assert py 40](#_Toc12277271)

[Tableau 5 - Exemple d'utilisation de Junit 5 41](#_Toc12277272)

[Tableau 6 - Résultat de l'exécution d'un kata, Junit 5 41](#_Toc12277273)

[Tableau 7 - Avantages et inconvénients de la containerisation de l'exécution. 42](#_Toc12277274)

[Tableau 8 - Mesures, Exécution containerisée vs hostée 43](#_Toc12277275)

[Tableau 9 - Payload d’un JWT 50](#_Toc12277276)

[Tableau 10 - Liste des états d'un kata 66](#_Toc12277277)

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’en a été épargné. Certain de ces domaines disparaissent progressivement, certains arrivent à tenir le cap malgré la concurrence de l’automatisation, d’autre métier meurt pour donner naissance à de nouveau métier. Mais dans le cas de l’apprentissage scolaire, il a fallu s’adapter pour survivre, au risque de devenir obsolète, car, encore obligatoire aujourd’hui, l’apprentissage scolaire est un piliers de notre société. 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 tel que DojoHepia au sein de l’enseignement.

Aujourd’hui, beaucoup d’outils sur le Web ont vu le jour, tel qu’Openclassroom, CodinGame ou encore Codewars. Ces outils, bien que très différents les uns des autres, sont des mines d’or d’apprentissage et de connaissance. De plus, ils sont tous des concurrents directs à l’apprentissage scolaire, 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.

Il m’a donc été proposer de créer cette nouvelle plateforme performante d’apprentissage en ligne, « DojoHepia ».

DojoHepia tient son inspiration des arts martiaux japonais, et notamment du Shōtōkan-ryū (c’est d’ailleur de là que la plateforme tient son vocabulaire). .Une notion relativement connue en programmation est le « kata ». En informatique, le kata est un exercice qu’un programmeur se doit de répéter souvent pour ne pas perdre les basiques d’un langage. Par exemple, il doit chaque mois créer un système pile avec pointeur en C, pour que cela deviennent un automatisme. De cette manière, le programmeur ne perd pas de temps sur ces notions lorsqu’un projet lui est donné. Cette notion de « kata » a fortement orienté ma plateforme à porter son vocabulaire sur les art-martiaux japonais et notamment Le Shōtōkan-ryū (cette aussi pourquoi DojoHepia contient le mot « dojo », lieu où l’on pratique les arts martiaux).

**Parler des chapitres**

# Avant-propos

**Cadre du projet**

Ce projet a été réalisé dans le cadre du projet de bachelor en ingénierie des technologies de l’information spécialisation logiciels et systèmes complexes, pendant la période scolaire d’avril 2019 à juillet 2019.

**Vocabulaire**

DojoHepia étant une application qui se veut proche du système des arts martiaux, Mr Cavat et moi même avons décidé d’y intégrer un vocabulaire propre à la plateform, qui hérite de différentes branches des arts-martiaux (Le Shōtōkan-ryū. notamment).

[TABLEAU DES TRANSLATION]

**But du projet**

Le but est de créer une plateforme d’apprentissage de programmation en ligne, destiné à Hepia et son infrastructure. Le projet s’appel “Dojo Hepia” en référence au Dojo, qui est une salle ou l’on pratique des arts martiaux, et Hepia, pour le nom de l’établissement. Cette Plateforme, spécialement construite pour l’apprentissage, permet aux Sensei de publier des programmes sur des points importants de la programmation (tel que les listes, les pointeurs ou encore les tableaux), et de les scinder en une série de katas.

Dojo hepia se voit donc doté d’un outil de programmation en ligne qui permet à un Utilisateur de de tester son raisonnement en direct, et de valider un exercice de cette façon.

**Fonctionnalités**

* Connection / Inscription
* Programme
  + Créer
  + Modifier
  + Supprimer
  + Dupliquer
* Kata
  + Créer
  + - Markdown
  + - Document
  + Modifier
  + Supprimer
  + Désactiver
* Participer à un kata
  + Essayer
  + Réussir
  + Abandonner
* Support des langages Python, MySQL et Java
* Recherche de programme (simple et avancée)
* Historique des katas
* Abonnement
* Sponsoring (Sensei)

Premièrement, il faut prendre en compte que même si ce service porte le nom de “compilation”, celui-ci est aussi un service d'interprétation car il permet d'interpréter du code Python. Pour simplifier la lecture de ce rapport, je parlerais désormais “d'exécution” pour en généraliser son fonctionnement.

**EXPLIQUER QUE BEAUCOUP DIMAGE NE CONTIENNENT PAS LE MENU DE GAUCHE POUR DES SOUCIS DE VISIBILITE EXPLIQUER DAND UN CH COMME MINELI**

Certaine figure (comme la fig n), ne sont pas représentative du site de manière global, ces figure ont pu subire des rognage, des modifications de taille.

**faire une partie avec les mocks frère**

à travers un service de gameification blahblah, parler des bienfaits pédagogique, avec la réussite etc.. l’exp utilisateur blahblah

**Bénéfices utilisateurs**

Elèves :

Professeurs : Il peut simplement publier une série ou plusieurs séries d’exercice sans se tracasser à suivre les blahblah

**Cahier des charges**

???

**Méthodologie de travail**

Le travail de bachelor est un travail de 3 mois avec une moyenne de 40 heures par semaines (généralement entre 9h et 18h). Dans le cadre de ce travail, 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” (le scrum est une méthodologie de travail, divisée en étape, et constituée d’itération).

**parler des bonnes pratiques de travail pour la reprise du projet**

**Le SCRUM BOARD**

Le développement travail était séparer en itération, ce qui a permis de choisir les fonctionnalités que j’allais implémenter à chaque itérations. 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 “SCRUM BOARD”, qui est un tableau physique, rempli de post-it, triés et catégorisé.

[IMAGE DU TABLEAU - PREMIER]

Le tableau se sépare en plusieurs parties :

BACKLOG

TODO

WORK IN PROGRESS

* ATTENTE DE VALIDATION

DONE

d’une itération type

Lorsque je devais commencer une nouvelle itération, dire qu’on se retrouvait avec cavat, que on validait les anciens post-it, qu’on reflechissait aux prochaines fonctionnalité à implémenter (utiliser le terme “pipe-line de developpement)

Versionning

|  |  |  |
| --- | --- | --- |
| Nom de version | Date de publications | Ajout |
| DojoHepia-v0.1 | 8 Mai 2019 | * Serveur de compilation   + Docker     - Support Java     - Support python |
| DojoHepia-v0.2 | 9 Mai 2019 |  |
| DojoHepia-v0.3 | 15 Mai 2019 |  |
| DojoHepia-v0.4 | 16 Mai 2019 |  |
| DojoHepia-v0.4.5 | 16 Mai 2019 |  |
| DojoHepia-v0.5 | 29 Mai 2019 |  |

**parler de l’intrication des trois autres personnes**

Comme Mr Cavat couvrait le projet de trois autres personnes, chacune d’elle a eu le droit au même traitement, à savoir l’obligation de développer avec une méthode dite agile. Cela m’a permis d’avoir 3 personnes avec qui converser de mes problèmes lors de réunions hebdomadaire que nous organisions tous ensemble. Aussi, Mr. Melvin Gay était mon testeur attitré, ce qui m’a permis d’avoir des retour fréquent pendant le développement de DojoHepia, et de ce fait, améliorer considérablement la plateforme (un avis extérieur est toujours très rafraîchissant).

**Faire un listing de tous les fichiers présent dans le rendu final.**

Chapitre 1 : Analyse fonctionnel

# Analyse du marché et des besoins

Pour que DojoHepia devienne une bonne application, 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, tel que Codewars ou encore CodinGame (ces deux exemple sont loin de se ressembler), et quelques autres plateformes, qui sont plus orientées apprentissage général (tel qu’openclassroom) plutôt que entraînement, c’est d’ailleurs pour cela que nous n’aborderons pas ces plateformes, car rappelons le, l’objectif de DojoHepia est d’être une plateforme d’apprentissage de programmation ou 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’aborderais ces points à travers l’analyse des sites Codewars et CodinGame, qui sont les sites internet qui se rapprochent le plus de ce que DojoHepia à pour volonté d’être.

## Codewars

Codewars à déjà la notion de dojo et de kata, que je cherche à faire apparaître sur ma plateforme. Au contraire de DojoHepia, le site est axé sur un système de compétition ou 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 kata, plus il monte en grade. Dans le Judo traditionnel et ses dérives, 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 sont les moyens officiels d’imposer une hiérarchie, mais les élèves sont plus habitués à recevoir des ceintures. Lors de la réalisation avec succès d’un kata, l’utilisateur gagne des points, qui permettent de faire évoluer un utilisateur dans un tableau des scores, et c’est la que se trouve tout l’aspect compétitif de Codewars.

L’interface est très chargée et il est très compliqué de comprendre ou s’orienter lorsqu’on arrive la première fois sur le site. Les informations importantes ne sont pas mises en avant et cela à 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 clan 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 niveau avec le rappel aux arts-martiaux (les Kyu et les Dan) et 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 adaptée à certains katas, non pas qu’il est moins bon qu’un autre utilisateur (très représentatif de la mentalité du Judo par exemple). La distinction est minime, mais pourtant très importante. De plus, cette plateforme comporte un système très intéressant de commentaire. En effet, à la manière de GitHub, les utilisateurs sont capables d’ouvrir des “Issue” (à traduire par “Problèmes”), qui leur 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 lorsqu’on abandonne ou que l’on réussi un exercice, l’utilisateur à 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.

## CodinGame

CodinGame est un site d’apprentissage en ligne axé jeu-vidéo. Le but est de résoudre des puzzles ou des niveaux de jeux en écrivant soi-même les lignes du code. Comme Codewars, et malgré un système d’aide lors de l’arrivée d’un utilisateur, l’interface 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 plateforme comme un moment ou 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 indispensable mais qui n’en reste pas moins intéressante, est que l’utilisateur peut changer le thème du site entre Light et Dark (lumineux et sombre), ce 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’ami, qui permet de comparer son avancement par rapport à eux, et de visualiser leur succès (Cela induit un petit système de compétition).

## Points relevés

Comme l’analyse de Codewars et CodinGame la 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 important devront tôt ou tard être implémenté pour DojoHepia pour rendre la plateforme encore plus fonctionnel et plus agréable à utiliser. Notamment, un système général de clan, et de tableau des scores, ou 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 meilleure que les autres équipes, car l’objectif de notre plateforme est d’être l’apprentissage avant tout.

Monter en grade implique un système de point gagné et d'obtention de grade, qui permet de se rapprocher d’un outil ludique ou l'apprenant apprécie passer du temps. Ce système de modification permettra de limiter les créateurs de kata, qui pourrait être limité aux utilisateur ayant le grade minimum requis.

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

# Game-ification

Chapitre 2 : Conception

# Préambule

Dans ce sous-chapitre, je présenterais l’architecture général de DojoHepia, sa structure de base de données, ainsi que les technologiques et les motivations qui m’ont poussées à les utiliser.

# Architecture

DojoHepia est doté d’une architecture dîtes “4 tiers” ;

* Service Client, *aka Client* (front-end)
  + Angular
* Service Serveur de traitement, *aka Gateway* (backend)
  + Javalin
* Service Serveur de compilation, *aka Service Compilation (backend)*
  + Javalin
* Service Base de données NoSQL (data provider)
  + MongoDB

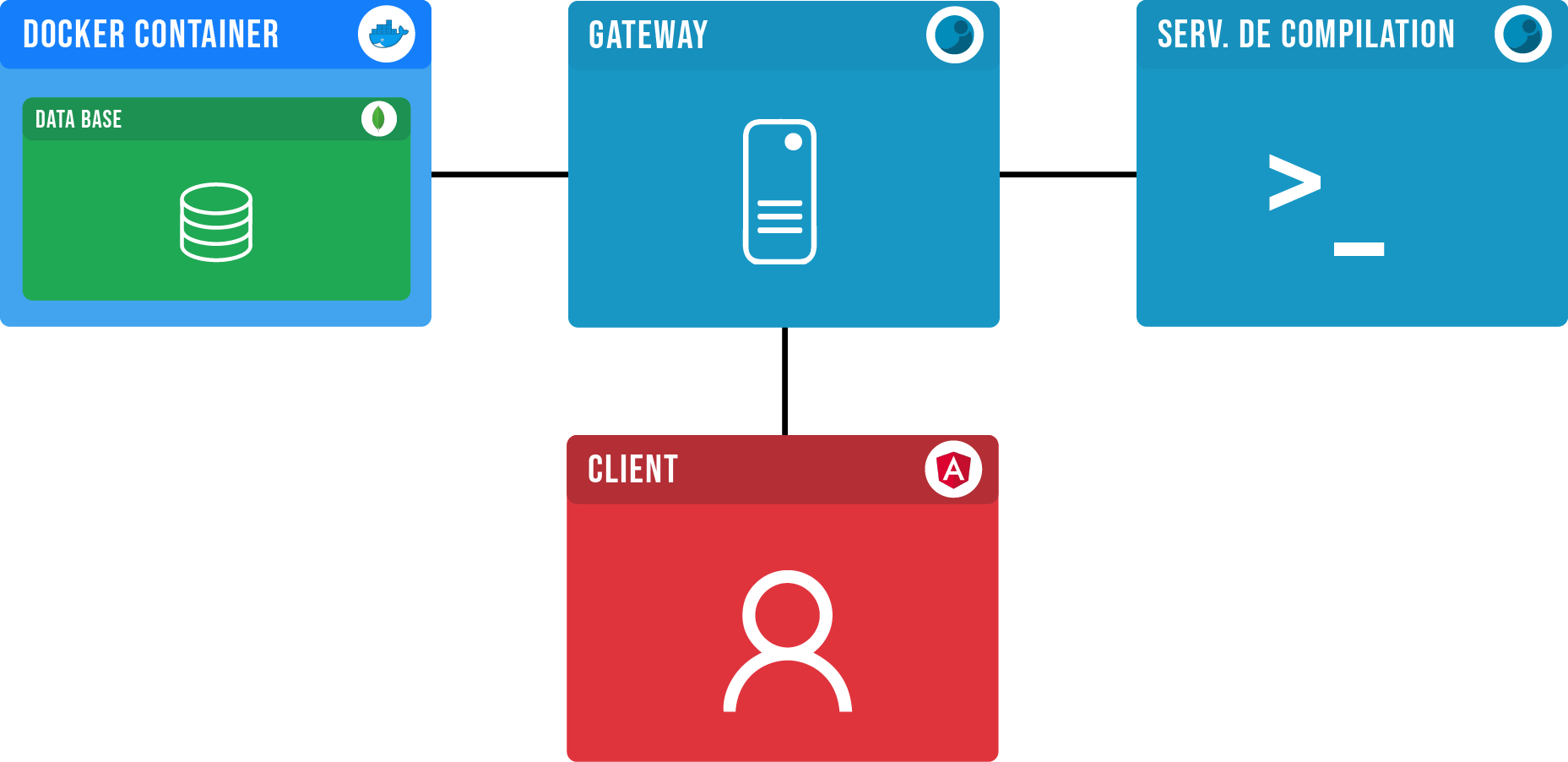


Figure - Architecture de DojoHepia

## Client

Le client est l’interface de DojoHepia, il a été conçu avec Angular (v7). Il permet de communiquer avec le Gateway, et d’afficher le contenu de la base de données.

## Gateway

Le gateway est le lieu de rencontre de tous les services, il est implémenté avec Javalin, qui est un Framework pour faire un serveur HTTP en Java. C’est avec ce service que le client peut demander les informations d’un programmes ou encore exécuter un kata. De plus, le service est « stateless », c’est à dire que la réponse d’une requête ne dépend pas du moment où elle à été envoyée.

## Compilation

Le service de compilation (implémenté avec Javalin) permet de compiler/interpréter du code reçu par le Gateway, et de renvoyer le résultat au Gateway.

## Data Provider

Le service de base de données, fonctionnant avec MongoDB, permet de stocker et faire des traitements sur les données de DojoHepia. (DOCKER ??)

## Vision à long terme

Le Gateway et le service de Compilation ont été conçus pour être scalables dans une vision à long termes.

*« 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 serveur différents. Une telle séparation des services permet de les remplacer facilement. Une des volonté de la plateforme, est, dans le future, 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 enverrai ses tâches d’exécution dans un pipeline de progression, 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) quanta lui, pour s’occuper des tâches d’exécution, et en restituer le résultat dans un pipeline de réponse.

[SHEMA DU WORKERSYSTEM]

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

# Choix technologiques

Mes motivations pour ce projet se trouvent aussi dans le choix technologiques. J’avais pour volonté de sortir de ma zone de confort et de me confronter à l’apprentissage de nouvelles technologies. Cela me permet 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 préférer (NodeJS, RethinkDB, etc..) pour en explorer de nouveaux. De plus, pour pouvoir créer une application robuste, les conventionnel NodeJS et autres Framework se basant sur du javascript n’était pas de mise, car javascript est un langage non typé (les variables n’ont pas de type).

## Angular

Pour ce qui est de la grande tendance web du moment, c’est Angular qui est sur le devant de la scène. Ce Framework simple et à la fois complet permet de développer un front-end simple 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 (grâce aux “component” et “services” notamment). C’est aussi pour moi le moyen d’aborder un nouveau Framework. La simplicité de mise en place combiné avec un puissant outil créé par nos amis de chez Google, en font un outil de confiance et de qualité.

## Javalin

Javalin 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.

## MongoDB

Parmis les nombreux choix de base de données, c’est MongoDB qui a retenu mon attention, c’est pour moi un nouveau langage, qui se rapproche des langages non relationnels que j’ai utilisé dans le passé (Notamment RethinkDB). Sur le marché du travail, c’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 très documenté (le langage utilisé dans le MongoSHELL est du javascript, cela rend son utilisation d’autant plus facile).

Atomic ?

Docker ?

# 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, <https://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.

## Users

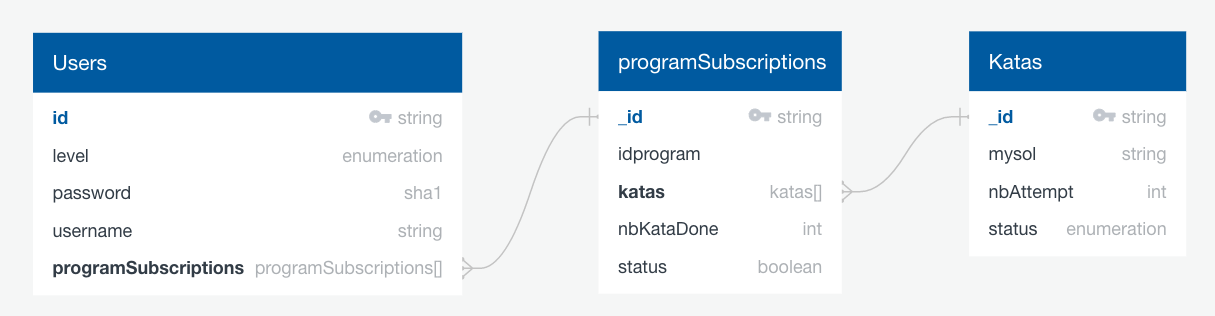
**

Figure - La collection Users et ses sous-collections (AV, 2019)

**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, 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 simples[[1]](#footnote-1), et permettent de stocker un utilisateur avec le minimum d’information nécessaire

Sous-collection - programsSubscription

programSubscribtion permet à l’utilisateur de s’abonner à un programme et d’y sauvegarder sa progression. Le champs nbKataDone permet d’indiquer le nombre de Kata qu’un utilisateur a effectué. Status quanta 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.

REVOIR LES CARDINALITE POUR LES IMAGES DE LA BDD

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[[2]](#footnote-2) passe de “TODO” à “ONGOING”. La solution de l'utilisateur une fois le kata terminé, (qu’il soit abandonné ou réussi) sera stocker dans le champs “mysol”.

## Programs

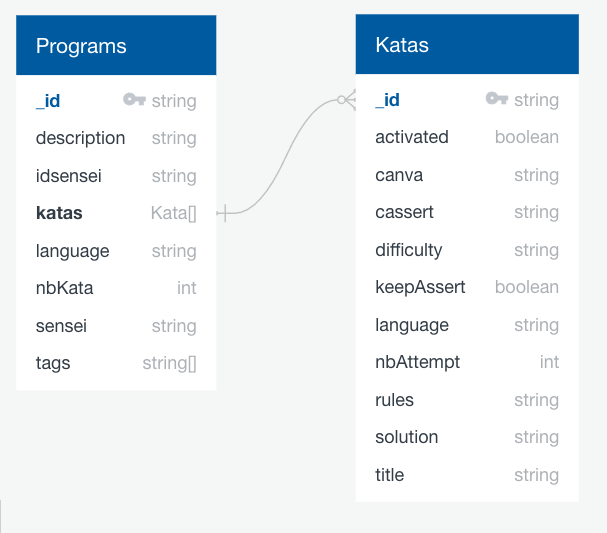
**

Figure - La collection Programs et sa sous-collection (AV, 2019)

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”. Celle-ci permet de stocker toutes les informations nécessaire à la description d’un programme. Le champs Katas est une sous-collection que je détail dans “”.  Quant au champs tags, il permet de stocker sous la forme d’un tableau les tags qui catégorise un programme, cela est notamment utiliser pour filtrer les programme (voir partie kfeajd).

*Sous-collection - Kata*

La sous-collection Kata permet de stocker les informations nécessaire pour décrire ceux-ci. Les Champs “canva” “cassert” “solution” sont le stricte minimum pour la réalisation d’un exercice, ils permettent respectivement de stocker la base de code qui sera donnée à l’utilisateur lors de sa tentative de résolution du kata, de stocker la batterie tests du kata, et de stocker la solution de l’exercice, dans le cas où l’utilisateur abandonnerai. Une petite particularité, est le champs “keepAssert”, il permet de spécifier si le créateur du kata souhaite que la batterie de tests soit visible à l’utilisateur lors de la réalisation d’un kata.

Chapitre 3 : Mise en œuvre

# Préambule

Ce chapitre porte sur la mise en œuvre général de DojoHepia, il permet d’aborder comment les différentes parties que constituent la plateforme ont été conçus,

# Le Gateway

# Services Angular

Les services Angular sont des classes qui permettent de créer une interface de communication avec le Gateway. Ils donne la possibilité de questionner celui-ci, pour recevoir / poster / mettre à jour ou encore supprimer des informations. On peut catégoriser ces services en 5 parties :

* + - Les programmes
      * Interactions avec un programme
      * Les abonnements aux programmes
      * La recherche de programmes
    - Les katas
      * Interaction avec un programme
      * Les abonnements aux kata
    - La compilation
    - L’authentification
      * Connection, inscription, génération de token
    - Le service LANG

Le but de ce sous-chapitre est d’en donner la liste complète, et de décrire le rôle et les utilités de chacun. Pour des questions de lisibilité, voici un tableau qui décrit les abréviations utilisées dans la description des services :

|  |  |
| --- | --- |
| 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 |

## Les programmes

|  |  |  |
| --- | --- | --- |
| Nom du service (.service.ts) | Méthodes | Description |
| Interactions avec un programme | | |
| program | create(obj) | Créer 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) | Créer l’abonnement d’un utilisateur à un programme |
|  | getMine(uid) | Récupère tous les programme créer 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 champs « type » correspond à « res » |

## Les katas

|  |  |  |
| --- | --- | --- |
| Nom du service (.service.ts) | Méthodes | Description |
| Interactions avec un kata | | |
| kata | publish(k,pid,isGoal) | Créer 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 |
| Kata-subscription | Create(obj) | Créer la première sauvegarde d’un utilisateur sur un kata |
|  | Increment(obj) | Augmente le nombre d’essai 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 |  |

VOIR AVEC CAVAT SI CEST PAS TROP MAM

# 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 - Vue des programmes (AV, 2019)

Les programmes sont des ensembles de katas créé par un Sensei, auxquels un utilisateur peut s’abonner et y sauvegarder sa progression. Ils sont orientés sur un langages de programmation (ou type de kata) en particulier, ce qui oblige son créateur à ne poster que des kata correspondant au langage du programme. De cette  manière, le créateur est obligé de séparer les kata par langages et non par notions.

|  |  |  |
| --- | --- | --- |
| Par notions | Par langage | |
| Programme - Les listes   * + - Les listes en Java     - Les listes en Python | Programme – Java   * + - Les listes | Programme – Python   * + - Les listes |

Tableau - Les programmes par notion / par langage

Les programmes sont agencé en “carte” (éléments de bootstrap, voir FIG N). Toutes ces informations (FIG N) permettent d’orienter l’utilisateur de manière optimal lorsqu’il s'agit de choisir un programme

|  |  |
| --- | --- |
| https://lh3.googleusercontent.com/gmZZe9pWP-52wxBKYzkLc04yJ1cTaH8n3A_uh0jSRS6sacNLb8ikeROuRoK7SNuNBWQjaQnL8_O7eGc_DD-3gC9xKILaS6rNlIJS4q-f_qvHjUnD6_GAl8DRMiMFnd0rCBNcOBU-  Figure - Carte d'un programme (AV, 2019) | 1. Titre 2. Nom du Sensei 3. Langage 4. Description 5. Tags 6. Nombre de kata |

.

## Implémentation

* **Missing diagramme mettre le diagramme de classe, avec les fonctions qui ne concerne que les diagramme, JE SAIS PAS ENCORE OU METTRE**

## Filtrer un programme

Les cartes programme ont été conçu pour être filtrable, pour que l’utilisateur puisse s’y retrouver avec la multitude de programme présent sur la plateforme. En effet, il est possible d’appliquer un filtre (Voir figure N) sur, le nom du Sensei, le langage du programme et ses tags.



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

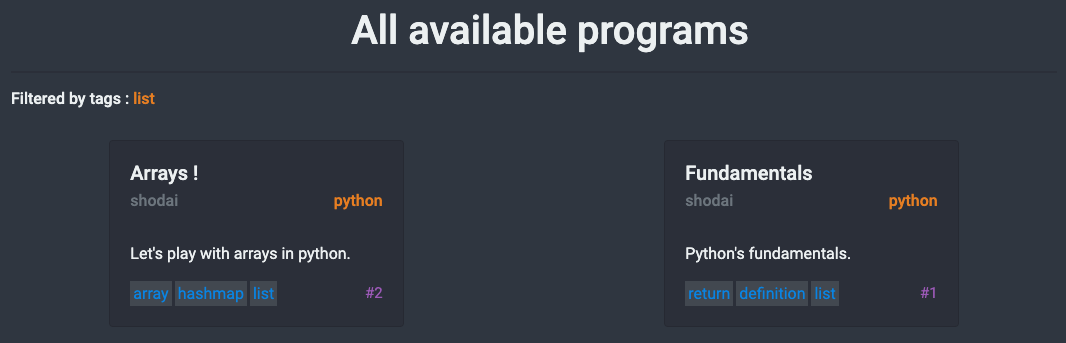


Figure - Vue des programmes, Filtre sur tag "list" appliqué (AV, 2019)

Filtre sur la page : Tous les programmes

Dans le cas de la page “tous les programmes”, à chaque fois que l’utilisateur clique sur un champs de filtre, le client va demander au Gateway de lui fournir une nouvelle liste de programme comportant les programmes correspondants au filtre souhaité. Le filtre n’est pas directement appliqué sur les programmes déjà récupéré, car dans le cas où le site contiendrait un nombre de programme conséquent, il ne serait pas tous rapatrié sur la page d'accueil (pour des questions de charges et de temps de traitement). Ce qui permet donc de filtrer sur le contenu présent dans la base de données, et non le contenu limité de la requête effectuée au préalable pour afficher les programmes.

Filtre sur les pages : Mes programmes & Mes abonnements

En ce qui concerne les pages “Mes programmes” et “Mes abonnements”, comme les programmes sont en quelque sorte déjà triés, le filtre lors d’un clique utilisateur s’effectue sur le contenu de la requête effectuée lors du chargement de la page.

## Recherche

J’ai mis en place un petit outil de recherche, pour que l’utilisateur puisse rechercher les programmes sur leur titre.

## Gestion

Le créateur d’un programme se voit doté de 4 outils de gestions d’un programme, permettant de rendre leur utilisation la plus complète et intéressante possible.

Créati

Edition

Suppression

Duplication

# Les katas

Les kata sont le centre névralgique de l’application, c’est sur eux que s’est portée la plus grosse partie du développement. Ce chapitre portera donc sur les katas, de la création de ceux-ci, à leur réalisation.

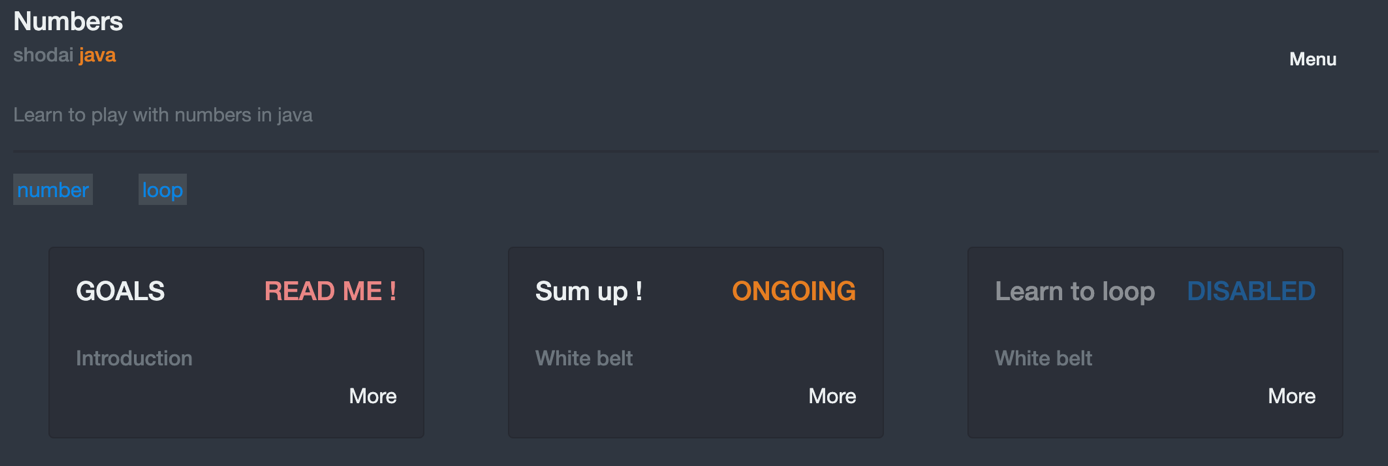


Figure - Vue des katas, Monji (AV, 2019)

Les katas sont des exercices créés par un Sensei, auxquels un Monji ou un autre Sensei peut participer, ils permettent de faire évoluer la progression d’un utilisateur dans un programme, en les réalisant. De plus, ils sont eux aussi agencé sous forme de carte (voir figure N), pour maintenir une cohérence dans l’affichage des ressources à travers la plateforme.

|  |  |
| --- | --- |
| Figure - Uu kata, utilisateur Sensei (AV, 2019) | 1. Titre 2. L’état 3. Niveau de difficulté 4. Menu du gestion, si authentifié en Sensei |

Il existe deux types de kata, les katas à réaliser, et les kata faisant le descriptif des objectifs du programme.

## Les états

Les katas se divisent en plusieurs états caractérisés d’un code de couleur pour les identifier rapidement (Voir Tableau N). Ils permettent d’indiquer simplement à l’utilisateur sa progression 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 à abandonner le kata, il débloque 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 à désactivé le kata (visible au Sensei seulement) | Le Sensei désactive un kata |

Tableau - Les differents status d'un kata

**implémentation**

* **Missing diagram JE SAIS PAS ENCORE OU METTRE Peut être avant introducion, faire un préembule de ce qui sera abordé dans le chapitre, puis balancer le diagramme de classe, puis interface puis implémentation**

## Interface de réalisation d’un Kata

Pour que l’utilisateur puisse réaliser un kata en ayant tous les outils en mains, j’ai dû imaginer une interface complète, mais simple (c’est dans ce genre de réflexion que l’analyse du marché s’est avérée très utile). L’objectif était de naturellement guider l’utilisateur sur la page, la lecture se faisant de haute en bas, et de droite à gauche, toutes les informations devaient apparaître dans un ordre cohérent.

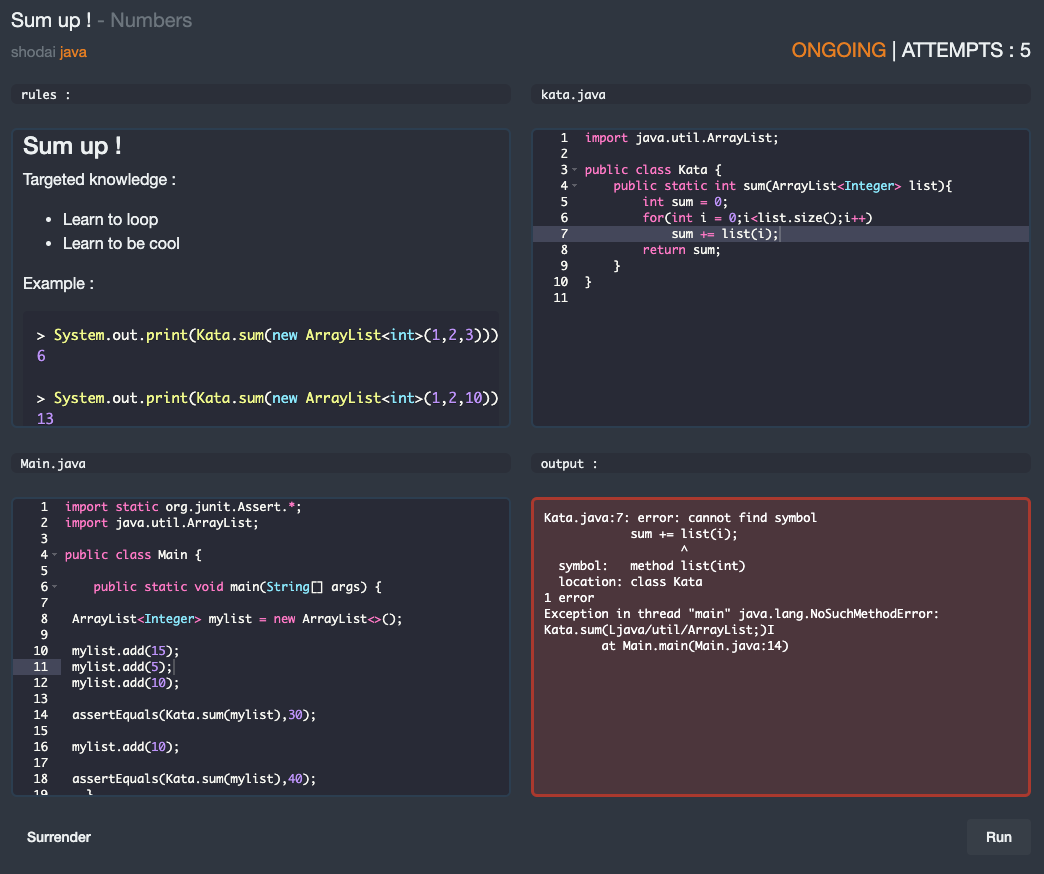
****

Figure - 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, à 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’essai déjà effectué.

Les outils de réalisation

1. Règles du kata (“rules” *dans FIG N*)

Cette partie permet de décrire l’objectif à réaliser pour le kata. Les règles peuvent-être écrite en texte pure, en markdown, mais peuvent aussi être une image ou un PDF .

1. Canva (“kata.java” *dans FIG N*)

Cette partie contient le canva (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.

1. Boîte de batterie de tests (*“Main.java” dans FIG N*)

La boîte de batterie de tests est une partie facultative de l’interface de réalisation d’un kata, le choix de la montrée ou non se fait dans la création d’un kata. Cette boîte permet d’afficher le code qui va évaluer la fonction que l’utilisateur a écrit (en somme, valider le prédicat que le Sensei a écrit (voir partie service de compilation)).

1. Boîte de sortie (*output dans FIG N*)

C’est dans la boîte de sortie que va être montré le résultat de l'exécution du code écrit par l’utilisateur (voir partie réalisation d’un kata).

Les boîtes de canva et de batterie de tests, découle d’un module d’édition de code en temps réelle, *ace editor*, ce module permet aussi la coloration syntaxique des langages. Il est donc très utile pour plonger l’utilisateur dans une bonne expérience d’outil de programmation en ligne.

Les options

J’ai décidé d’offrir à l’utilisateur un moyen d’abandonner le kata, pour débloquer la solution si il s’avérait trop difficile pour lui. Toutefois, il lui faudra réaliser un certain nombre d’essai avant de pouvoir débloquer la solution, nombre qui est choisi par le Sensei lors de la création du kata. Le bouton « Run » (Execution en francais), permet de tester son raisonnement, et de valider le kata si il s’avère juste.

Si l’utilisateur abandonne un réussi le kata, l’option « try again » (Refaire le kata), se débloque, et permet à l’utilisateur de refaire le kata, sans que la bonne réussite de celui-ci n’impacte à nouveau son score.

## Exécution du code

Pour que ma plateforme puisse remplir sa fonction d’apprentissage 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 orienter vers des outils de d’exécution en ligne, tel que jdoodle (*JDoodle - free Online Compiler, Editor for Java, C/C++, etc* 2019). Malheureusement, les outils de ce type sont généralement payant et fonctionnent avec un système d’abonnement (ou alors contiennent des version dites “d'évaluation”, insuffisantes pour nos besoins). 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 de compilation/interprétation, et la solution adoptée est d'exécuter le code que l’utilisateur souhaite faire valider sur un service dédié.

## Le service de compilation

Ce service est un serveur Javalin ayant pour objectif générale de renvoyer le résultat du code que l’utilisateur souhaite valider. Pour pouvoir tester le code de l’utilisateur, j’ai dû trouver une solution pour que l’utilisateur puisse lui-même créer une batterie de tests. La solution qui a tout de suite été retenue est d’utiliser les librairies 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

Qui permet de vérifier si le résultat de la fonction donnée en paramètre correspond au résultat attendu, lui aussi donné en paramètre. Cela permet au Sensei d’évaluer la fonction qu’il souhaite faire valider. Voici les librairies d’assertions utilisées pour l’implémentation des langages :

* Python : Assert Py
* Java : Junit

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) |
| **def** mult(arr,n):  **return** [i\*n **for** I **in** arr] | **from** assertpy **import** assert\_that  **import** sample **as** m  assert\_that(m.mult([2,4,6],3)).is\_equal\_to([6,12,18]) |

Tableau - Exemple d'utilisation d'assert py

|  |  |  |
| --- | --- | --- |
| Temps [s] | Résultat | Code d’erreur |
| Exemple : Exercice 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 - 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 cette exercice, le but est de sommer tous les éléments d’un tableau donné.

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

Tableau - Exemple d'utilisation de Junit 5

|  |  |  |
| --- | --- | --- |
| Exemple : Exercice 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 - Résultat de l'exécution d'un kata, Junit 5

Avec ces librairies, il est donc possible de proposer au Sensei de composer un exercice 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.

## Implémentation

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 machine virtuelle fonctionnant sur le kernel hôte, (*Docker* 2019)), pour éviter des problèmes de sécurités 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 | | Execution 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 déployable | Facilité de créer un container par implémentation de langage |  |
|  |  | La maintenance d’une image est très simple, et très modifiable |  |
|  |  | Comme le container est supprimé après son utilisation, y effacer des fichiers ou corrompre son contenu ne posera pas de problème |  |

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

Fig N

Comme le montre la figure N, 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 fig. N), mais c’est un mince sacrifice pour les avantages que nous apporte 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 - Mesures, Exécution containerisée vs hostée

Un des points importants relevé dans les avantages de la containerisation (voir fig. N), 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 contenait les outils nécessaires à l’exécution des langages Java et Python.

L’image Python

FROM python  
RUN pip3 install assertpy  
RUN mkdir /env/

Listing - Dockerfile de l'image python

Comme le montre le Dockerfile (fichier permettant de construire une image, voir <https://docs.docker.com/engine/reference/builder/>, (*Docker documentation - Dockerfile* 2019)) du Listing 1, l’image python est basée sur ‘python’ qui est une image déjà présente sur Dockerhub ([www.hub.docker.io](http://www.hub.docker.io), (*Docker Hub* 2019)), Cela permet d’avoir une image python stable et complète. Il suffit juste d’ajouter la librairie d’assertion. La commande « RUN mkdir /env/ » créer un répertoire à la racine de l’image nommé « 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 - Dockerfile de l'image java

Cette fois-ci, le Dockerfile (voir Fig N), 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 (*Java SE Development Kit 11* [sans date]), qui lui, me permet de compiler et exécuter le langage java. Il se charge aussi d’ajouter le fichier .bashrc (voir Listing N), 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 - 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 deux fichiers, « java\_test.sh », qui est une petite suite de commande bash (voir Listing N), permettant d’effectuer des actions utiles avant de pouvoir correctement utiliser le container, et le .jar de junit.

#!/bin/bash

source /root/.bashrc

javac -classpath ${CLASSPATH} Main.java Kata.java # Compile files and librairies

java Main # Run the application

Listing - 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 stockent 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 (*Use volumes* 2019). <- METTRE LA SOURCE LINK

Lancement des containers, docker run

Voici une description des 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 - 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 - Commande docker, permettant de créer et lancer un container java

Options communes :

-- 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 N), lance l’interprétation du fichier « assert.py », pour java (voir listing N), lance l’exécution de « java\_test.sh ».

## Communication entre les différents services

La figure N présente le workflow de communication entre les différents services

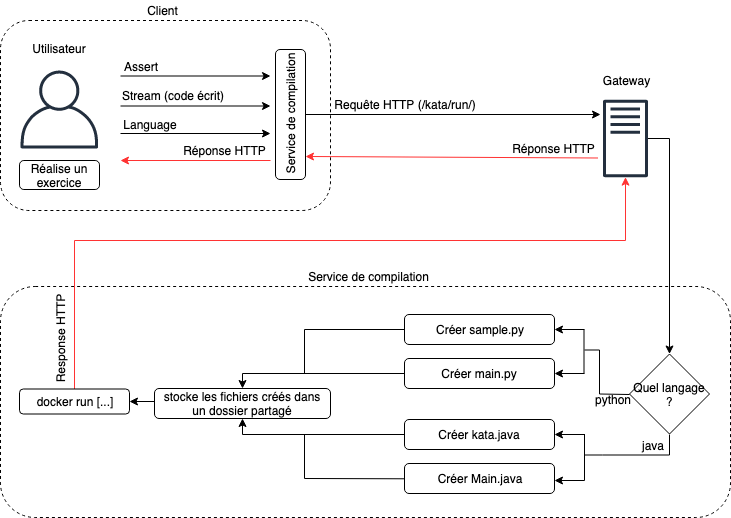
****

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

Contenu du body lors de la réception de la question du Serveur

La requête http envoyée du Gateway au service de compilation se présente comme ceci :

|  |  |  |
| --- | --- | --- |
| **Requète http** | | |
| **Nom du champ** | **Type** | **Contenu** |
| stream | string | Contenu du code de l’utilisateur |
| assert | string | Batterie de tests |
| language | string | Langage cible |

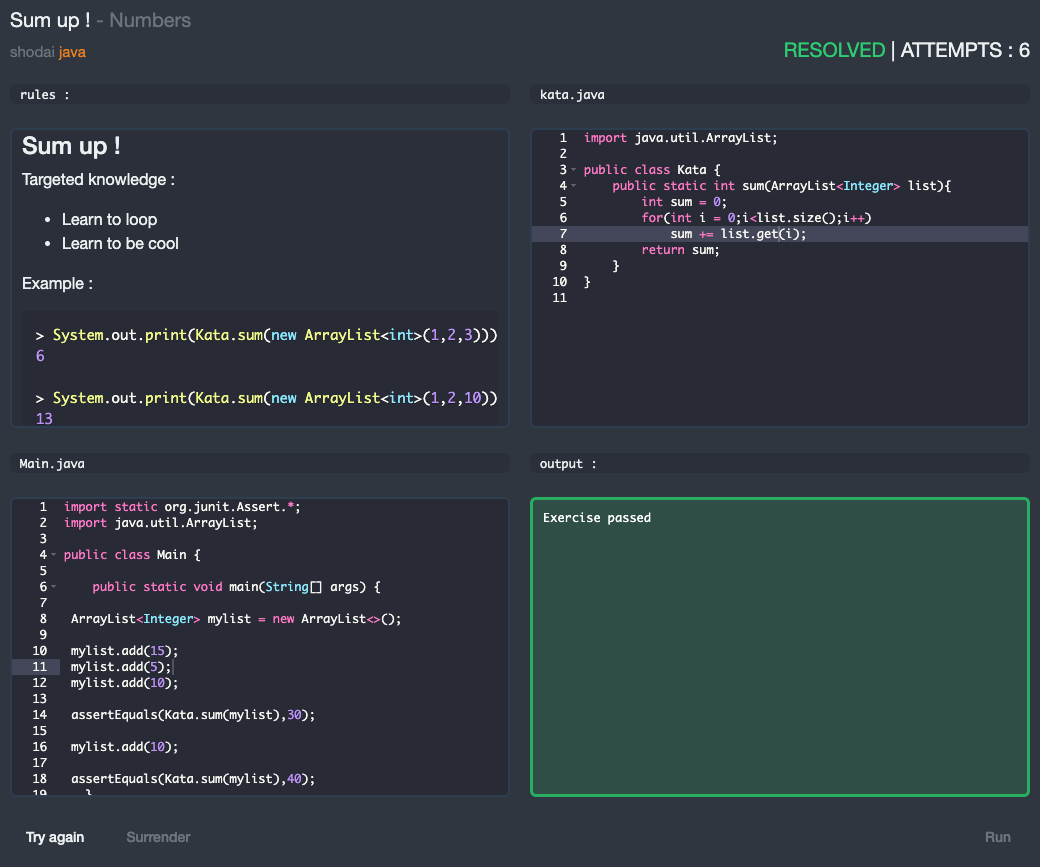
**Contenu du body lors de l’envoi de la réponse du Serveur**

La réponse http envoyée du service de compilation au Gateway se présente comme ceci :

|  |  |  |
| --- | --- | --- |
| **Requète http** | | |
| **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 |

## Réussir un kata

Dès lors que l’utilisateur à valider un kata, son statut se met à jours pour devenir “RESOLVED” (Résolu) et la solution de l’utilisateur ainsi que son nombre de kata effectué se mettent à jour (à noter : un kata effectué est soit résolu, soit abandonné).

****

## Abandonner un kata

Si un utilisateur veut abandonner un kata, il doit d’abord atteindre le nombre d’essai indiquer par le créateur lors de la création du kata. Une fois que celui-ci aura atteint ce nombre, il sera en mesure d’abandonner et de débloquer la solution. Si l’utilisateur décide d’abandonner, il débloque la solution et le statut du kata devient alors “FAILED” (échoué).

*A partir de ce moment, l’utilisateur, qu’il ait réussi ou abandonner, peux reessayer le kata, sans que sa progression n’en soit impactée.*

## 

## Gestion

Comme pour les programmes, le Sensei se voit offrir de petits outils de gestion pour ses katas (voir Figure N), en voici la liste :

Edition

Desactivation

EXPLIQUER QUE SA DECREMENTE LE NOMBRE DE KATA  DONE POUR TOUS LES RESOLVED ??

Supression

Créer un Kata

dire que la partie de l’execution sera abordée dans le chapitre (chapitre) plus bas.

**Parler de l’interface ? et de Ace editor et du markdown etc ??**

**ici, parler du service LANG**



Figure - Gestion d'un kata, menu déroulant

# Gestion des ressources et privilèges

Comme vu dans la partie “Architecture web”, j’ai choisi d’implémenter un service de type sans états (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.

## JWT

*Explications tirées du site officiel de JWT* (*JSON Web Tokens - jwt.io* 2019) *. Le but de cette introduction est de faire l’analogie avec mon implémentation.*

JWT, ou “JSON Web Token” est un standard d’échange sécurisé d’information au format JSON. Les informations transmises avec JWT sont fiables, et ce parce qu’elles peuvent être signés avec un secret, ou une paire clé privée / clé public (RSA ou ECDSA). Nous le verrons un peu plus loin, mais j’utilise un secret pour générer les tokens.

Un JWT a trois composantes :

* Header
* Payload
* Signature

Et tout assemblé, ressemble à ca :

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.

eyJleHAiOjE1NTkwNDM4MTh9.

EiUc5iYOzw8C6JLGUts7Eemm2C7pZhrrEpEhmYGbWjA

Listing - Un JWT

Header

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.

Le header 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 8 - Header d’un JWT

Payload

eyJleHAiOjE1NTkwNDM4MTh9.

Le payload est quant à lui composé de “Claims”, qui sont des informations, concernant de manière générale notre utilisateur. Dans mon implémentation, le payload lors de la génération du token 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 - 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 - Exemple du payload d’un JWT

Signature

EiUc5iYOzw8C6JLGUts7Eemm2C7pZhrrEpEhmYGbWjA

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 à grâce à l’algorithme spécifié dans le header.

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

## Implémentation

Le client et le Gateway sont dotés d’un système de rôle, 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 avec 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

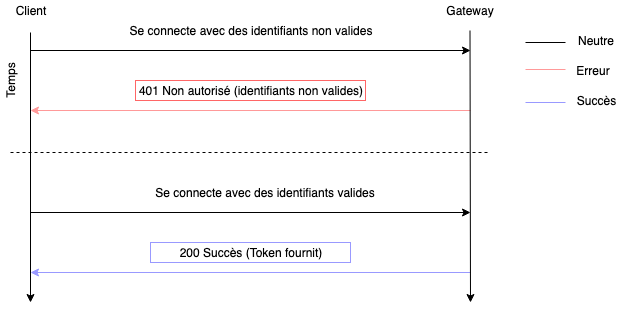


Figure - Authentification d’un client au serveur (AV, 2019)

Comme présenté dans le schéma ci-dessus, 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 par Client sont stockées dans LocalStorage. Le LocalStorage permet de stocker de manière persistante et locale des données sous la forme <clé ;valeur>, et ce, même si l’utilisateur ferme son navigateur.

Accès à une ressource

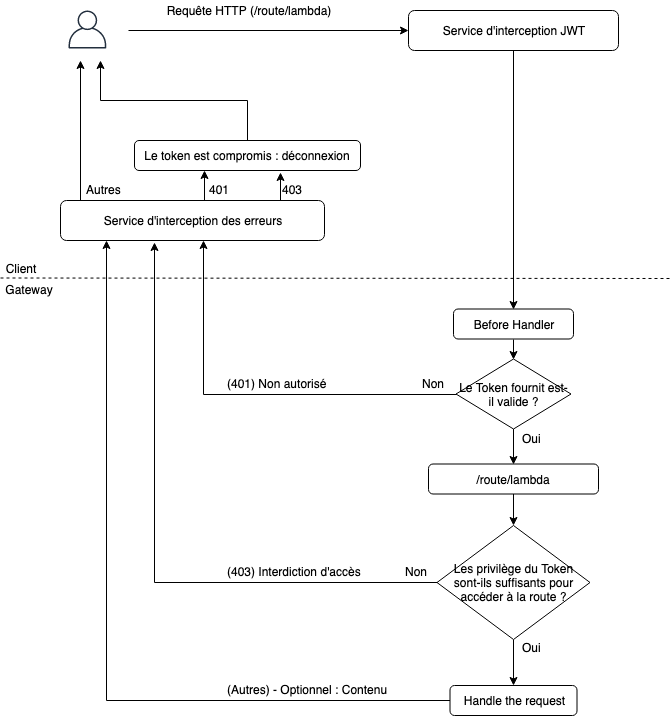


Figure - Cycle de vie d’une requête http (AV, 2019)

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 accompagné d’un Token. Pour ce faire, j’ai mis en place un service d’interception de requête http.

Angular Services – JWT Interceptor

Ce service permet d’intercepter chaque requête émise, et mettre le token stocké dans le LocalStorage dans les headers. Ainsi, cela nous permet de ne pas surcharger tous les autres services.

A partir de là, le serveur peut nous répondre de trois manières différentes :

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

Angular Services – Error Interceptor

J’ai aussi mis en place un service d’interception des erreurs, ce services colle un objet de type “Observable” [https ://angular.io/guide/observables](https://angular.io/guide/observables)  A nos requête émises, et lorsque celles ci reçoivent une réponse du serveur, le code d’erreur reçu est intercepté par ce service. Cela nous permet de centraliser tous les retours de requête 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 de LocalStorage (fait office de déconnexion), ce qui compromet son Token).

En ce qui concerne le serveur, chaque route est précédé de ce qu’on appel un “before handler” (un petit bout de code qui s’exécute avant le code principale de la route). Ce handler s’assure que l’entête de la requête contient bien un Token valide. Cette validité dépend de plusieurs facteurs :

1. Le Token est-il contenu dans les en-tête 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 désiré, 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 notre cas, une erreur de type 403 ne devrait pas arriver si l’utilisateur utilise le site de manière conventionnelle, car les accès sont aussi gérer 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 renvoi le contenu avec un code de réponse (200)

**Accès à page route Client**

*Angular Guard – Authentification*

Lorsqu’un utilisateur souhaite accéder à une route, un “Authentification Guard” (qu’on pourrait apparenter au *before handler* du Gateway) regarde si l’utilisateur remplit les conditions nécessaire pour y accéder. Il y a deux conditions :

Être connecté

Avoir le role 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 respecté, l’utilisateur est renvoyé à la page d’accueil.

De cette manière, nous nous évitons des problèmes ingérables d’accès aux ressources non autorisé (ex : l’utilisateur accède à une route qui lui est restreinte, et envoi une requête avec des privilège insuffisant au serveur).

# Les abonnements

Pour pouvoir gérer de manière efficace quelles utilisateurs suivaient un programme, nous avons mis en place un système d’abonnement aux programmes, qui permet les choses suivantes :

* Garder un suivi de son évolution sur un kata (Nombre d’essai, solution, etc..)
* Garder un suivi de son évolution sur un programme spécifique
* Protéger un programme d’un mot de passe
* Avoir une page “Mes abonnements”
* …
* …
* …
* ..

Expliquer comment marchent les abonnements, comment l’utilisteur y save sa progression

**Logiciels utilisé par DojoHepia**

|  |  |  |
| --- | --- | --- |
| Nom | Description | Licence |
| Angular | Web Framework | MIT |
| assertpy | Librairie d’assertion pour python | BSD |
| Bootstrap | Librairie d’interface utilisateur | MIT |
| Brace | Librairie contenant de code couleur pour ace editor | MIT |
| Java |  |  |
| Javalin |  |  |
| js-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 |
| Ubuntu |  |  |
| uuid | Librairie générant des ID uniques | MIT |

Conclusion

Améliorations ?

Extensible

* Système de game-ification (ranking, team-up, accès à de nouvelle fonctionnalités, par ex : proposer un exos)
* Un système de commentaires sous un exercices
* L’utilisateur ne pourra visualiser le résultats de l’exercices qu’après multiples essais (n) (peut-être en échange de points de ranking ?)
* l’utilisateur peut visualiser l’historique de ses exercices
* Implémentation de nouveaux langages (compilation support)
* Leaderboard

Conclusion

ANNEXES

Annexes 1 : Tests et Résultats

annexe : test effectués

1. Acceder au site sans un token valide
2. Acceder à des ressources sans un token valide
3. Acceder a des routes sans les privilège reqcuis
4. Créer un kata avec un contenu conséquent (> 8096 Byte)
5. Acceder à la réalisation d’un kata sans être abonner a son programme
6. Acceder a un programme qui n’existe pas
7. Modifier un programme qui n’est pas le siens
8. Modifier un kata qui n’est pas le siens

Annexe 2 : manuel de l’utilisateur

Ce manuel a pour but de vous aider à trouver vos marques sur DojoHepia.

# Sensei

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

En tant que Sensei, vous avez la possibilité de :

* Créer un programme
* Gérer votre programme
* Créer un kata
* Gérer votre kata
* Visualiser vos programmes
* Sponsoriser un nouvel utilisateur

## Créer un programme

## Gérer votre programme

## Créer un kata

## Gérer votre kata

## 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 appel « Le sponsoring ». Sur la page de sponsoring, vous avez le choix de générer un token pour un Monji ou un autre Sensei, et de choisir le temps avant le token ne soit expiré.

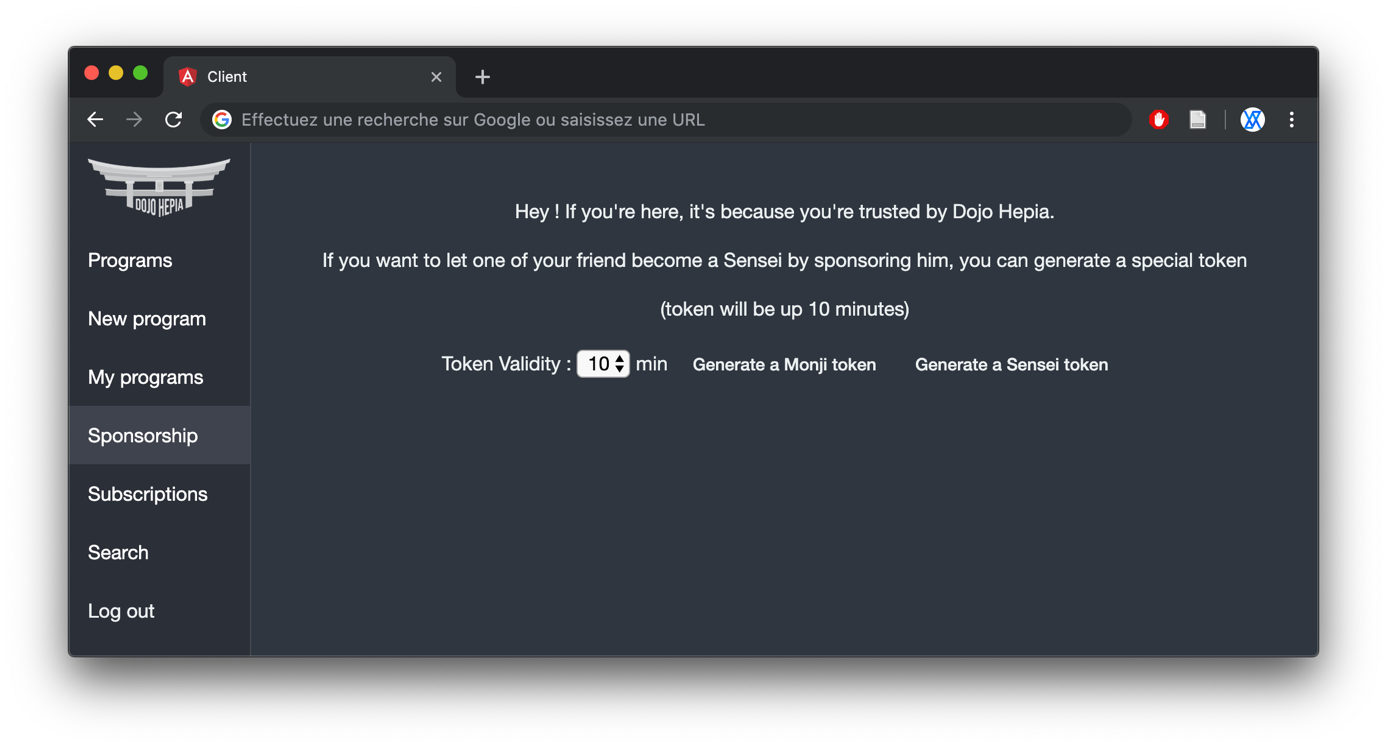


Figure - Manuel utilisateur : Sensei - Page de Sponsoring (AV, 2019)

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 apparaitra en rouge (voir figure Figure 16), et vous pourrez le copier pour le donner à un utilisateur qui voudrait rejoindre DojoHepia.



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

# Monji

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 FIG N).



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

Filtrer un programme

Si vous cliquez sur le nom du Sensei ((n) sur figure N), ou sur le langage ((n) sur figure N), ou sur un des tags ((n) sur figure N), cela vous permettra de filtrer le programme par l’entité cliquée.



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

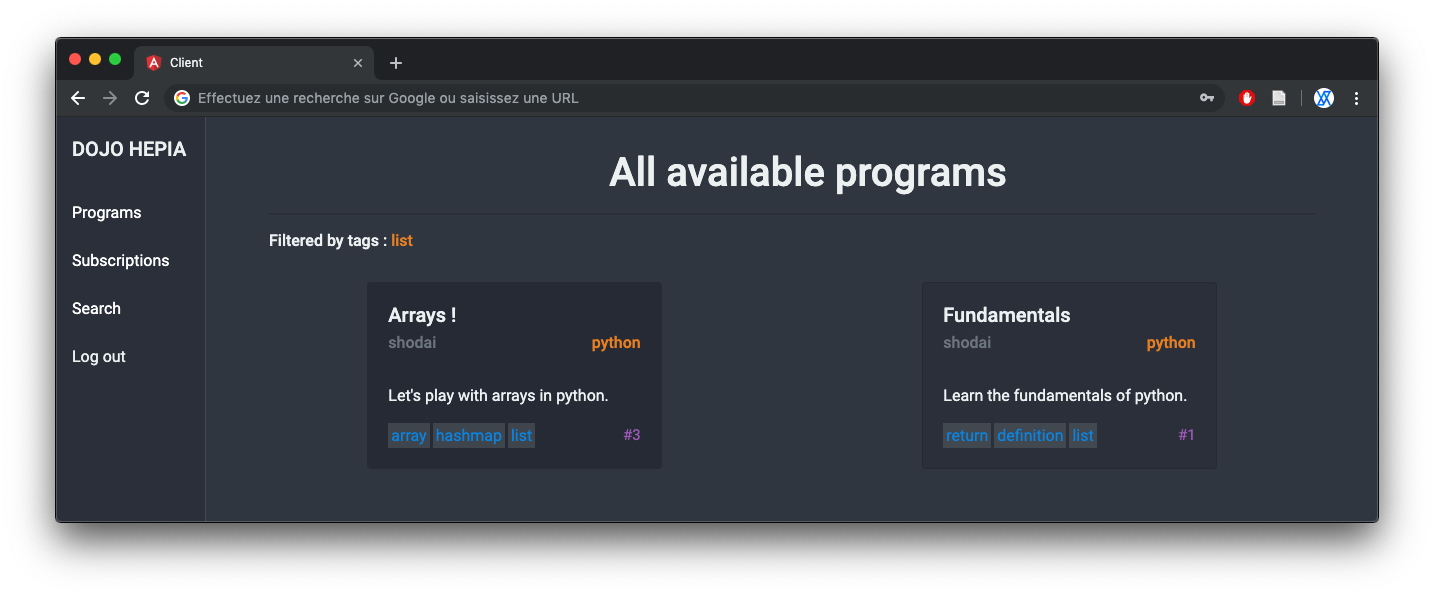


Figure - Manuel utilisateur : Monji -Page : Tous les programmes disponibles | Filtrée par tag (AV, 2019)

## Afficher un programme

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

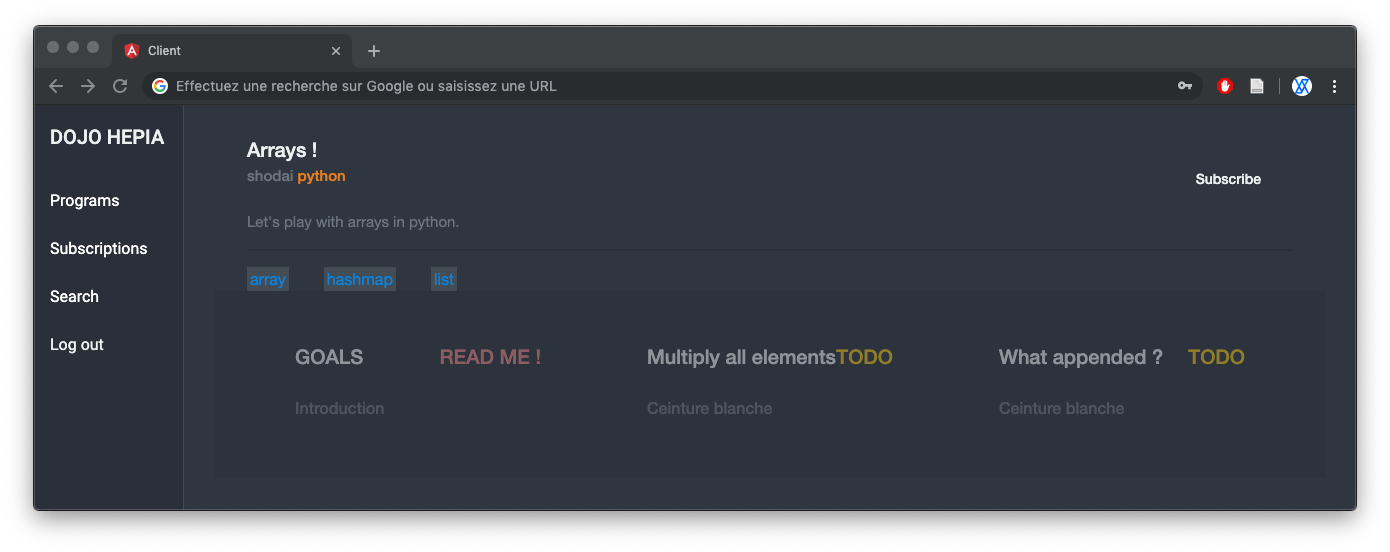


Figure - Manuel utilisateur : Monji - Page d'un kata, utilisateur non-abonné (AV, 2019)

En premier lieu, si vous n’êtes pas abonné au programme, les katas vous apparaîtront désactivé (voir FIG n), il faudra vous abonner au programme (voir section : n'importe qui - s’abonner à un programme) pour pouvoir les réaliser.

Si vous êtes abonnée au programme, il vous apparaîtra comme dans la FIG N. Dans le bandeau d’entête, les informations affichées (titre, sensei, langage, tags) sont similaire à celles d’une carte programme (voir FIG N).

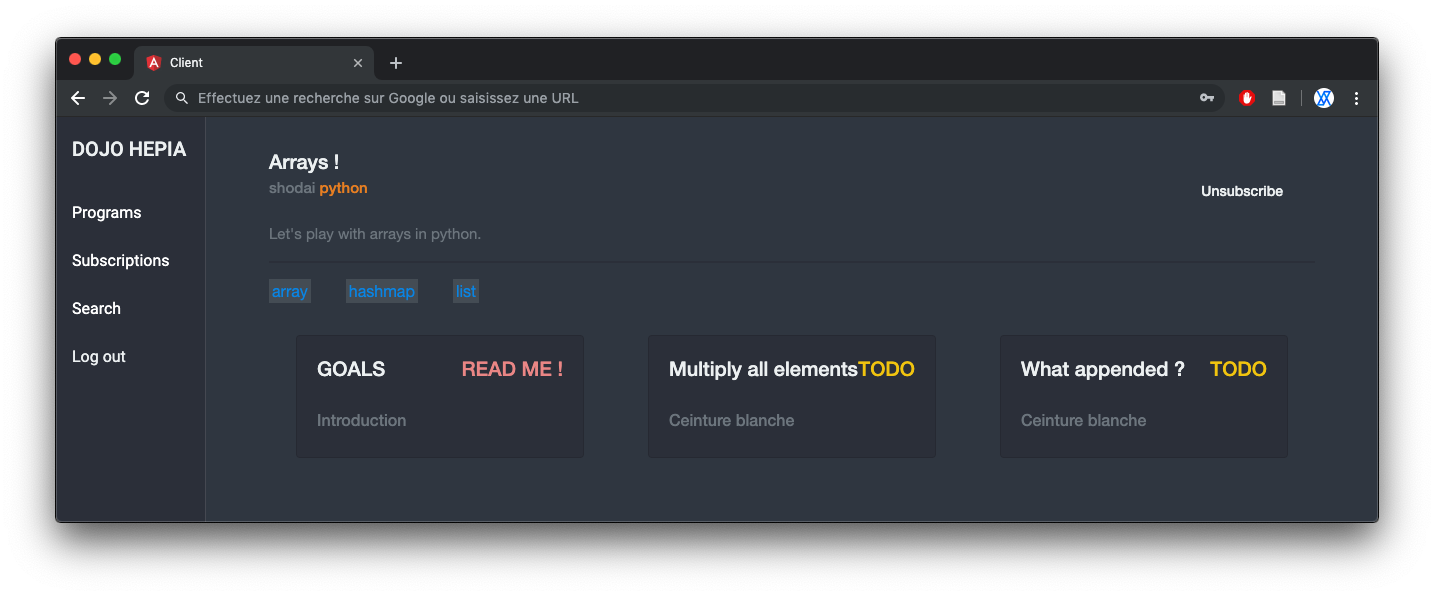


Figure - Manuel utilisateur : Monji - Page d'un kata, utilisateur abonné (AV, 2019)

sur cette vue, vous pourrez voir votre progression sur les différents katas proposés. Les programmes sont composé d’objectifs, ils portent le titre “GOALS”, et doivent-être lu avant de réaliser un kata. chaque kata / objectif à un état de progression, la figure N 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 débloquez 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 |

Tableau - Liste des états d'un kata

## Vous abonner à un programme

Si vous souhaitez sauvegarder votre progression ou réaliser un kata sur un programme, il vous faut obligatoirement vous abonner à celui-ci. les programmes auxquels vous êtes abonnée ne vous apparaissent pas désactivé (voir plus haut FIG N) et sont visibles sur la pages “Mes abonnements” (voir plus bas dans séction ...).

Mot de passe

Certain programme nécessite un mot de passe, vous ne pourrez-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 désabonner d’un programme, il n’est plus visible dans la page “Mes abonnements”.

Si vous progressez dans un programme, vos avancements seront sauvegardé même une fois que vous vous désabonner 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

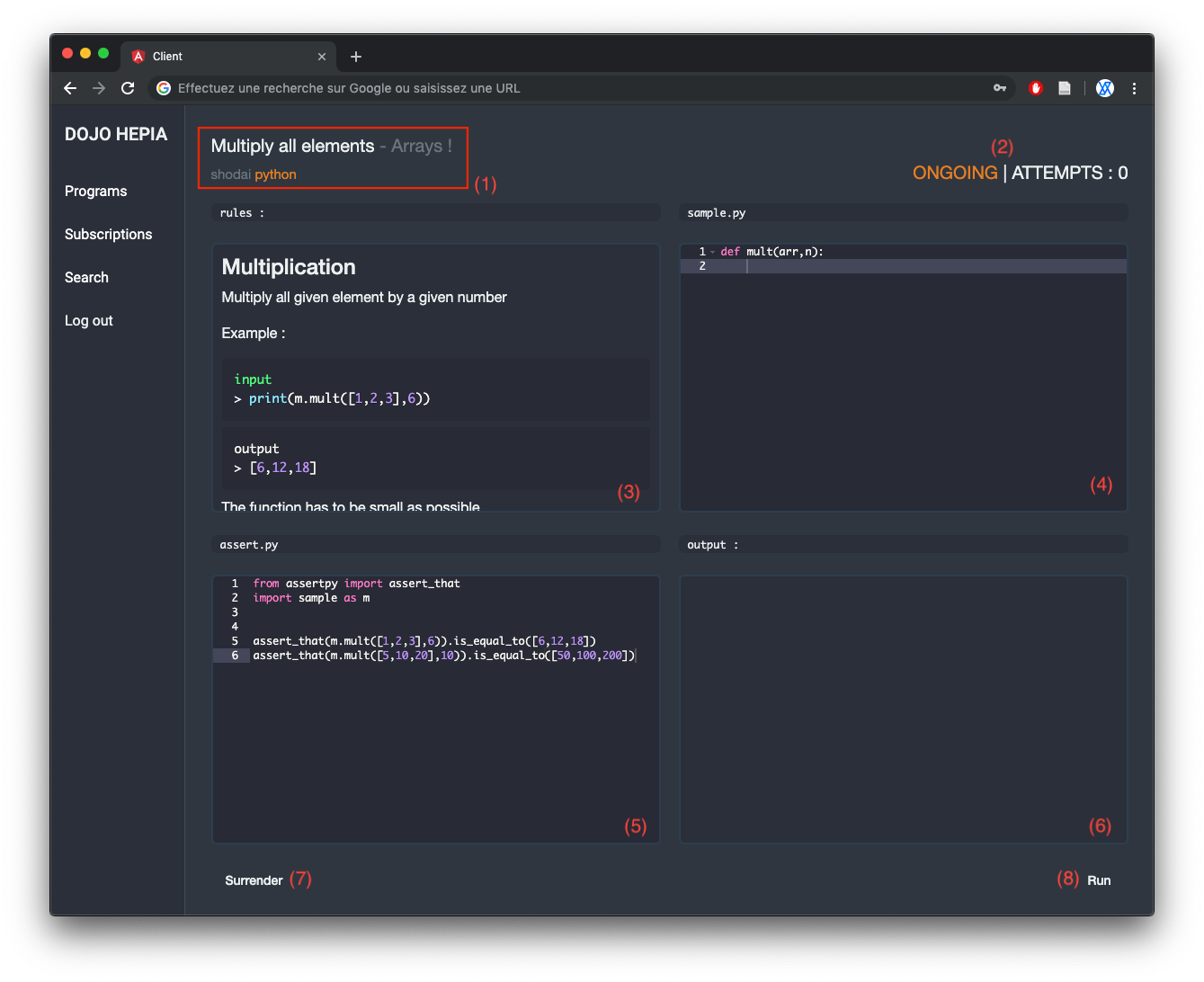


Figure - Manuel utilisateur : Monji - Interface de réalisation d'un kata

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’essai effectué
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 test
7. le bouton d’abandon, disponible après le nombre d’essai atteints, spécifiés 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 donnera pas de points en plus.

## Rechercher des programmes

La recherche d’un programme s’effectue sur 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 :

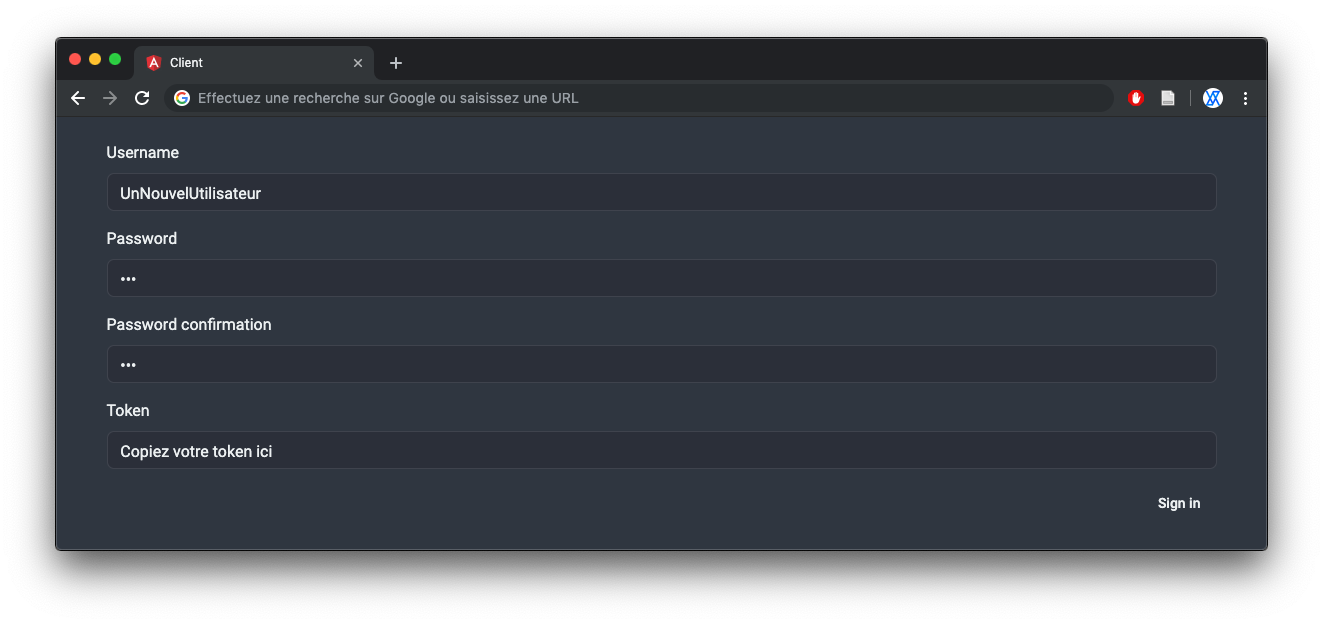
****

Figure - 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’exacte dans “Password confirmation”

Token

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

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 - Manuel utilisateur : N'importe qui - Page d'authentification

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

Annexe 3 : 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 partie :

* + 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 problème 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 stricte nécessaire dans votre image. Veillez à n’intégrer que les outils dont vous avez réellement besoins, 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 système d’assertion, mais tous ne sont pas forcément adapté 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 couche 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 10 - Dockerfile, python

L’image se base sur python, installe une librairie externe pour la batterie tests (assertpy), puis créer un répertoire « env » à la racine de l’image, qui, dans le cadre de python, et pour respecter une cohérence avec les autres images déjà créées, est indispensable pour que lors de la construction du container, la commande « docker run » puisse correctement attaché le répertoire partagé au répertoire partagé de votre machine hôte.

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 Dockerfile 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 (Listing N), car python n’a pas besoin d’être compilé pour être exécuté, et 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 nécessaire pour que le container s’exécute de la manière souhaitée. Une chose importante toute fois, c’est que votre flow d’exécution générale ne contient que deux fichier (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 N.

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

« fichier\_code.ext » : C’est le fichier ou est écrit le code tapé par l’utilisateur.

« fichier\_batterie\_tests.ext » : C’est le fichier ou 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 écrit 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 principale, c’est dans celui-ci que sont importées (au besoin) les dépendances nécessaire 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 en 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 fichier 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\_canva.ts

Ce fichier (voir Listing N) contient les informations nécessaire pour la création d’un kata, à savoir :

* + id : L’id du langage (nom du langage, exemple : ‘python’, ‘java’)
  + assertCanva : Le canva de batterie de tests, qui est le bout de code donné au Sensei pour qu’il créer sa batterie de tests (avec l’importation des librairie concernées aussi)
  + codeCanva : Le canva dans lequel le Sensei va écrire sa solution
  + filename : Nom du fichier ou l’utilisateur va écrire sa solution (Seulement à titre d’affichage, visible sur l’affichage d’un kata)
  + assertname : Nom du fichier ou 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 - 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. 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 - 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 - 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 le publier. De bon 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
  + Les temps d’attentes entre l’appui du bouton « run » et le résultat ne sont pas excessifs
  + L’ajout de libraire interne au langage ne pose pas de soucis

## Publication

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

Annexe 4 : Utiliser DojoHepia

Cette annexe décrit comment lancer tous les services de DojoHepia, pour pouvoir utiliser la plateforme.

Références documentaires

1. Le champs « level » est évoqué dans le chapitre 4 dans la partie « Gestion des ressources et privilèges » [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)