Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES À FINALITÉ SPÉCIALISÉE EN DATA SCIENCE

CHATssistant

Un agent conversationnel pour l'apprentissage de la programmation

Dalla Valle, Maxime; Jacques, Antoine

Award date:
2020

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 27. Oct. 2026

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

CHATssistant

–

Un agent conversationnel pour l'apprentissage de la programmation

Maxime DALLA VALLE
Antoine JACQUES

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Remerciements

Nous adressons nos remerciements sincères à nos deux promoteurs, le Professeur Benoît Vanderose et le Professeur Benoît Frénay pour leur attention, leur aide et leurs conseils apportés tout au long de la réalisation de notre mémoire de fin d'études.

Notre reconnaissance s'adresse également aux membres de la faculté d'informatique de l'UNamur avec qui nous avons eu la chance d'échanger durant notre stage.

Nous remercions nos parents, nos frères et sœurs, Sarah pour leur soutien durant nos études et la réalisation de ce mémoire.

Résumé et abstract

Résumé

L'apprentissage de la programmation est un domaine de plus en plus étudié. Au fil des années, de nouvelles méthodes sont apparues pour donner un meilleur support pédagogique vis-à-vis de cet apprentissage. Dans ce mémoire, nous proposons une approche basée sur la qualité du code de l'élève. Dans le cadre du cours *[INFBO131] Introduction à la programmation*, nous avons développé un agent virtuel conversationnel, plus communément appelé chatbot, qui permet aux étudiants de lui poser des questions concernant la programmation. Celui-ci pouvant également donner un retour sur le code en remplaçant la console d'un IDE. Ce chatbot crée un lien avec l'équipe pédagogique du cours. En effet, celui-ci est capable de fournir rapidement des retours sur la qualité de code des étudiants. Dans notre solution nommée BOTLEARN, le chatbot est un outil utile aussi bien pour les élèves que le corps enseignant.

Abstract

Programming is an area that is increasingly being studied. Over the years, new methods have appeared to give a better pedagogical support to this learning. In this paper we propose an approach based on the quality of the student's code. Within the framework of the course *[INFBO131] Introduction to Programming*, we have developed a virtual conversational agent, more commonly called chatbot, allowing students to ask questions about programming. This one can also give feedback on the code by replacing the console of an IDE. This chatbot is the link with the teaching team of the course because it is able to give quick feedback on the quality of the students' code. In our solution called BOTLEARN, the chatbot is a useful tool for both students and teachers.

Table des matières

1 Introduction **8**
1.1 Contexte 8
1.2 Objectifs 8
1.3 Question de recherche 9
1.4 Plan du mémoire 9

2 État de l'Art **11**
2.1 Facteurs cognitifs 11
2.1.1 Styles d'apprentissage 11
2.1.2 Charge cognitive 11
2.1.3 Facteurs émotionnels 12
2.2 Facteurs psychologiques 13
2.2.1 Profils d'apprentissage 13
2.2.2 Fixed et Growth Mindset 14
2.2.3 Feedbacks positifs 15
2.2.4 Motivations 16
2.3 Difficultés et Misconceptions 18
2.3.1 Difficultés 18
2.3.2 Misconceptions 19
2.4 Les IDEs 20
2.5 Les différents langages 21
2.6 Solutions existantes 22
2.7 Chatbot 24

3 Conception **26**
3.1 Observations 26
3.1.1 Public observé 26
3.1.2 Analyse des résultats 27
3.1.3 Utilisation potentielle des résultats 29
3.2 Définition des bases du travail 30
3.2.1 Choix du langage 30
3.2.2 Choix de l'IDE 31
3.3 Solution proposée 31

3.3.1 Chatbot 32
3.3.2 Interface pédagogique 39
3.3.3 Serveur 42
3.3.4 Interactions de nos composants 49
3.3.5 Nom de la solution 54
3.4 Discussion 55
3.4.1 Pertinence des feedbacks 55
3.4.2 Maintenance et pérennité 55
3.4.3 Limites 55

4 Validation **59**
4.1 Méthodologie 59
4.2 Résultats 61

5 Travaux futurs **65**
5.1 Enrichissement de la base de données 65
5.2 Analyse du code 65
5.3 Statistiques 66
5.4 Gamification 66
5.5 Amélioration des Sandboxes 67
5.6 Poursuivre la validation 68

6 Conclusion **70**

7 Bibliographie **71**

8 Annexes **74**

Acronymes **79**

Glossaire **81**

Table des figures

1 Les différentes étapes rencontrées durant la réalisation d'un programme [Kinnunen and Simon, 2011]. 12

2 Classification en fonction des résultats [Carmo et al., 2006]. . . . 14

3 Déplacer un étudiant d'une mentalité à une autre [Murphy and Thomas, 2008]. 15

4 Les différents types de feedbacks [Kinnunen and Simon, 2010]. . 16

5 Exemples de difficultés et de misconceptions [Qian and Lehman, 2017]. 20

6 Pourcentage d'étudiants apprenant un certain langage de programmation et pourcentage de cours utilisant un certain langage de programmation [Mason and Cooper, 2014]. 21

7 Exemple d'utilisation de *Thonny*. 23

8 Exemple d'utilisation de *UUhistle* [Sorva and Sirkiä, 2010]. . . . 23

9 Exemple d'utilisation de *BlueJ*. 24

10 HelpDesk mis en place. 28

11 Infrastructure proposée pour répondre aux objectifs du mémoire. 33

12 Exemple de stockage des questions/réponses via un entraînement. En rose les questions/réponses entraînées et en jaune le stockage/matching en base de données [Cox, 2019]. 34

13 Exemple d'un cas d'utilisation du chatbot sur base d'un entraînement de questions/réponses. 34

14 Exemple d'une substitution de la chaîne de caractères "Saturday" vers "Sunday" en appliquant l'algorithme de Levenshtein [Trekhleb, 2018]. 35

15 Exemple de questions posées au chatbot par un étudiant dans le cadre d'une réalisation d'un exercice. 36

16 Exemple d'un feedback par rapport à l'exécution du code d'un étudiant. 37

17 Interface d'administration pour un exercice. 39

18 Questions posées par les étudiants ayant eu un matching. 40

19 Nombre d'erreurs par code d'erreur. 41

20 Architecture 43

21 Comparaison entre une requête avec ou sans cache 51

22 Récupération d'une réponse à la suite d'une question posée au bot 52

23 Sauvegarde d'un exercice par un utilisateur-administrateur. . . . 53

24 Exécution d'un exercice fourni par un utilisateur. 54

25 Interface d'un notebook [1] 56
26 Questionnaire en 26 questions. 60
27 Comparaison de BOTLEARN par rapport à 452 autres produits informatiques. 61
28 Résultats bruts des questionnaires. 62
29 Parties de notre architecture utilisées lors d'une requête de données non-présentes dans le cache. 74
30 Parties de notre architecture utilisées lors d'une requête de données présentes dans le cache. 75
31 Parties de notre architecture utilisées lors de l'exécution d'un code d'un étudiant. 76
32 Parties de notre architecture utilisées lors de la création d'un exercice. 77
33 Distribution par question par rapport aux résultats de la section 4.2. 78
34 Distribution de la moyenne par question par rapport aux résultats de la section 4.2. 78

1 Introduction

1.1 Contexte

L'apprentissage de la programmation est complexe et il le reste malgré la mise à disposition pour l'étudiant d'une multitude de solutions innovantes qui ont pour but de l'accompagner dans cet apprentissage. Sans être exhaustif, on peut évoquer des outils d'analyse statique de codes, des plateformes qui permettent d'observer pas à pas l'exécution du code ou encore des dispositifs avec de la "gamification".

D'année en année, le nombre d'élèves inscrits au cours d'introduction à la programmation n'a pas arrêté d'augmenter et il devient de plus en plus difficile pour les enseignants de satisfaire à la demande et d'octroyer un temps d'enseignement suffisant qui permet de rencontrer les attentes individuelles de ces étudiants.

La programmation résulte d'une construction en cascade de multiples concepts : si une étape n'est pas bien assimilée ou rédigée, la suite de la programmation sera un échec. Par exemple, la construction d'une boucle nécessite la bonne compréhension de la notion de condition sans quoi, le résultat de l'exécution du programme ne sera pas celui attendu. Tout au long de son apprentissage, il est important que l'élève dispose d'un outil pédagogique qui l'aide et lui apporte les solutions nécessaires lorsqu'il est confronté à un blocage ou une erreur. De même, l'enseignant doit pouvoir disposer des informations nécessaires sur les capacités des étudiants à progresser dans leur formation ainsi que sur les étapes difficiles qu'il convient d'éclaircir lors des formations pratiques individuelles.

Différents environnements de développement peuvent fournir un retour sur le code réalisé par l'étudiant, mais cela n'est pas optimal ; en effet, celui-ci reste souvent confronté à des questions non solutionnées ou à des réponses peu compréhensibles, ce qui crée un sentiment d'impuissance et peut mener à un abandon dans la démarche d'apprentissage.

Depuis quelques années, des agents conversationnels sont apparus, appelés chatbot, et utilisés dans divers domaines et métiers (support client, télémarketing, etc.). Ils permettent une interaction simple entre les systèmes informatiques et leurs utilisateurs.

Ces nouvelles techniques peuvent se révéler d'un intérêt majeur dans l'apprentissage de la programmation, en permettant à un étudiant non expert de disposer d'un soutien pédagogique directement disponible ainsi que d'une aide efficace dans la correction de l'exercice de programmation. Cependant, un outil fiable, efficace, aisé et disponible en français reste attendu.

1.2 Objectifs

Dans ce contexte, plusieurs objectifs ont été identifiés et vont accompagner la recherche de ce travail :

- proposer une aide efficace et motivante afin que l'étudiant puisse progresser dans son apprentissage de la programmation
- offrir un retour d'apprentissage à l'étudiant, constructif, compréhensible, et un accompagnement dans sa progression par des conseils et des renforcements positifs
- apporter une aide au corps enseignant par la création d'un outil utile dans les tâches pédagogiques et l'encadrement des étudiants qui suivent le cours d'introduction à la programmation.

1.3 Question de recherche

Pour les étudiants qui suivent le cours d'introduction à la programmation ainsi que pour leurs enseignants, quels sont les moyens réalistes et pédagogiques qui peuvent être mis à disposition pour apporter un encadrement efficace et permettre un meilleur apprentissage des techniques de programmation ?

- quels sont les facteurs qui influencent l'apprentissage de la programmation ?
- comment sont perçus les feedbacks donnés aux étudiants ?
- quels sont les types de feedbacks adéquats à proposer aux étudiants ?
- quelles sont les principales difficultés rencontrées par les étudiants dans leurs apprentissages ?
- quels sont les outils et les langages de programmation adaptés à l'apprentissage de la programmation ?

1.4 Plan du mémoire

La conception d'un chatbot pédagogique est la réponse apportée à la question de recherche formulée et ce résultat est le fruit de plusieurs étapes détaillées dans ce mémoire. Une telle solution s'est concrétisée au travers de la mise en place d'un serveur centralisé qui fournit un chatbot à destination des étudiants et une plateforme Web au service du corps enseignant.

Une recherche scientifique des éléments qui ont un impact sur la réalisation a été consignée dans le chapitre État de l'Art. Ainsi, des facteurs tels que la cognition, la psychologie de l'étudiant, les erreurs commises ont dû être gardés à l'esprit durant la mise en place de la solution. Des aspects plus technologiques comme les Environnements de Développement Intégré, les langages de programmation et l'analyse des solutions existantes ont déterminé les choix techniques de l'utilisation de l'agent conversationnel.

Dans la section 3.3 du travail, les différentes étapes de la réalisation du chatbot sont détaillées, depuis l'analyse des attentes des étudiants jusqu'au

choix du langage de programmation et de l'Environnement de Développement Intégré et finalement la description concrète et détaillée de la solution proposée et nommée BOTLEARN.

Finalement, l'utilisation du logiciel permettra de valider son caractère opérationnel et réaliste ainsi que de définir les évolutions possibles qui pourront lui être apportées.

2 État de l'Art

Ce chapitre passe en revue les informations nécessaires pour comprendre l'apprentissage d'un étudiant dans un cours de programmation. Les volets d'apprentissage cognitifs (section 2.1) et psychologiques (section 2.2) servent à comprendre le parcours d'apprentissage de l'étudiant et doivent être abordés dans un premier temps. Ensuite, il est nécessaire de s'attarder sur les difficultés (section 2.3) que l'étudiant peut éventuellement rencontrer dans son parcours. Le choix de l'Environnement de Développement Intégré (EDI) (section 2.4) sera également examiné. La pertinence du langage dans l'apprentissage de la programmation sera abordée (section 2.5). Puis, les solutions existantes pour faciliter l'apprentissage de la programmation (section 2.6) seront également explicitées avant de donner un aperçu des assistants virtuels (section 2.7).

2.1 Facteurs cognitifs

Cette section met en lumière certains facteurs ayant une influence sur l'apprentissage. Ces derniers permettent de mieux comprendre le fonctionnement des étudiants, et de saisir l'importance de la charge cognitive.

2.1.1 Styles d'apprentissage

Chaque personne a son propre style d'apprentissage et les étudiants qui suivent des cours de programmation n'échappent pas à la règle. De toute évidence, en l'absence de conseil ou guidance, les étudiants auront tendance à adopter le style qu'ils préfèrent ou qui les a le mieux servis dans le passé [Tandon and Ravikumar, 2013]. Deux styles d'apprentissage se dégagent. Il s'agit du **"deep learning"** et **"surface learning"**. Une personne "deep learner" aura tendance à comprendre la matière en profondeur tandis qu'un "surface learner" se concentrera plutôt sur les points clés. La difficulté de la programmation requiert un mélange de ces deux styles. Le "surface learning" est très utile pour se remémorer les règles de syntaxe d'un langage de programmation, mais développer des compétences en programmation nécessite un "deep learning" [Tandon and Ravikumar, 2013]. Les étudiants qui ont eu l'habitude d'étudier en surface auront du mal à se plonger dans le code tandis que ceux qui avaient pour habitude d'aller dans le détail auront du mal à avoir une vision globale d'un langage de programmation [Jenkins, 2001].

2.1.2 Charge cognitive

La charge cognitive est une théorie en psychologie qui s'intéresse à la capacité de stockage de l'information en mémoire de travail et à l'intégration de nouvelles informations. Elle tend à expliquer les échecs ou les réussites des personnes en situation d'apprentissage en général.

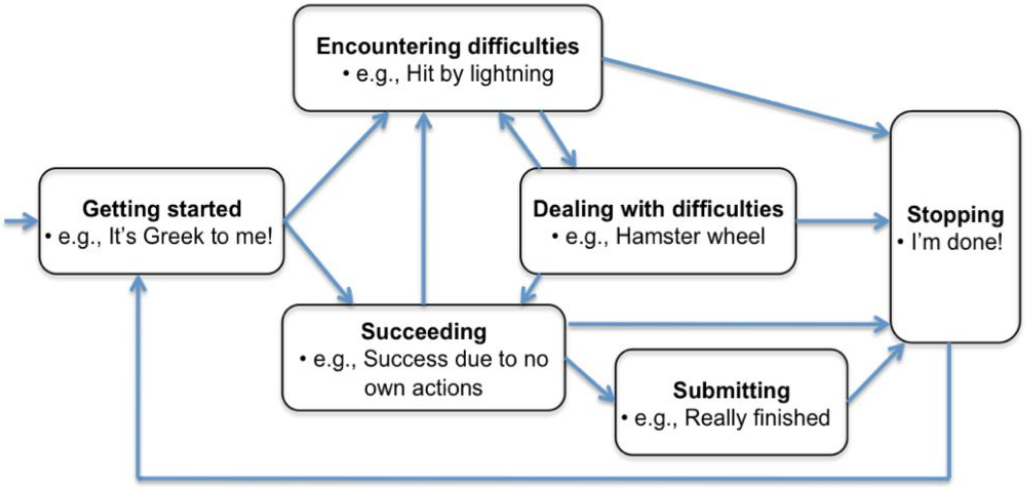


FIGURE 1 – Les différentes étapes rencontrées durant la réalisation d'un programme [Kinnunen and Simon, 2011].

Selon cette théorie, un être humain ne peut intégrer que sept éléments simultanés. Sa capacité d'apprentissage sera donc altérée lorsque le nombre total d'exigences dépasse sa capacité de mémoire (également appelé mémoire de travail). Toutefois, cette limite est appelée à varier selon les individus pour monter à neuf éléments intégrés ou bien être réduite à cinq [Mason et al., 2015].

Trois différentes sources de charge cognitive peuvent également être identifiées :

- **Charge intrinsèque** : il s'agit de la complexité de la tâche ;
- **Charge externe** : il s'agit de la clarté de l'information, comment l'information est présentée ;
- **Charge essentielle** : il s'agit de la concentration requise pour réaliser ou comprendre l'information.

Ces trois charges sont additives ce qui signifie qu'il serait théoriquement possible d'en avoir une « complète » (au niveau des sept éléments) et les deux autres complètement vides [Morrison et al., 2014]. La charge cognitive aura un impact significatif sur le choix de l'IDE (section 2.4).

2.1.3 Facteurs émotionnels

La réalisation d'un programme ou d'un exercice de programmation impose à la personne un cheminement par plusieurs étapes émotionnelles (figure 1). À chacune des étapes, l'étudiant est confronté à un ensemble d'émotions spécifiques. Par exemple, il semble naturel que dans l'étape "Encountering difficulties" les émotions comme la frustration ou la confusion soient fréquemment rencontrées [Kinnunen and Simon, 2010].

Les émotions ont un rôle prépondérant sur l'efficacité de l'étudiant durant l'apprentissage de la programmation. En effet, les émotions négatives ont une

corrélation négative dans l'accomplissement d'une tâche. Les phases "Getting Started", "Encountering difficulties" et "Dealing with difficulties" sont susceptibles d'être influencées par un grand nombre d'émotions négatives. Une attention toute particulière est nécessaire au niveau de ces trois phases pour atténuer les effets négatifs et garder l'étudiant dans une bonne dynamique [Kinnunen and Simon, 2010]. Il faut également souligner que le manque d'assistance à la suite d'obstacles rencontrés par les étudiants est un facteur de frustration supplémentaire et conduit à diminuer la qualité de leurs résultats [Bosch et al., 2013].

En lien avec la partie 2.1.2, le facteur émotionnel est également lié à la charge cognitive à laquelle il faut prêter attention. Pour désigner cette charge, le terme « charge émotionnelle auxiliaire » est utilisé [Kinnunen and Simon, 2010].

En Bref ! : Facteurs cognitifs

Trois points ont été abordés dans cette section. Premièrement, il y a le style d'apprentissage d'un étudiant, celui-ci pouvant être un "deep learner" ou un "surface learner". Deuxièmement, la charge cognitive est composée de trois sources différentes : intrinsèque, externe et essentielle. Troisièmement, il reste les facteurs émotionnels qui pourront être ressentis par un étudiant lors de la réalisation d'un exercice.

2.2 Facteurs psychologiques

Dans cette section, les facteurs psychologiques sont abordés. De façon non exhaustives, ceux mis en évidence dans cette section sont le profil d'apprentissage, de la mentalité, de la notion de feedbacks et de la motivation.

2.2.1 Profils d'apprentissage

Des psychologues ont démontré que chaque individu à sa propre façon de percevoir et d'interagir avec l'environnement d'apprentissage dans lequel il évolue. Un profil d'apprentissage peut être déterminé par cinq dimensions selon *Carmo* [Carmo et al., 2006] :

1. Sensoriel ⇔ Intuitif
2. Visuel ⇔ Verbal
3. Actif ⇔ Réfléchi
4. Séquentiel ⇔ Global
5. Inductif ⇔ Déductif

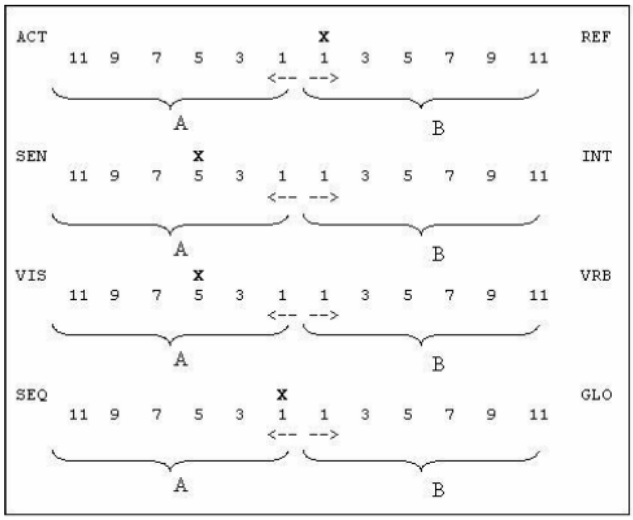


FIGURE 2 – Classification en fonction des résultats [Carmo et al., 2006].

Il est possible de déterminer le profil d'apprentissage via un questionnaire (figure 2). Toutefois, l'analyse du questionnaire n'est pertinente que si celle-ci est réalisée par un psychologue. Tout l'intérêt du profil d'apprentissage repose sur l'optimisation possible de la manière dont les contenus sont enseignés et les messages délivrés à l'attention des étudiants. Par exemple, pour la dimension numéro quatre, si un professeur sait qu'il enseigne à un étudiant avec une vision globale, alors il faudra présenter le problème de manière générale. À l'inverse, si l'étudiant est plutôt séquentiel, il faudra présenter le problème étape par étape [Felder and Soloman, 2011]. Le profil d'apprentissage est une notion importante, car il met en avant le fait que chaque personne a des capacités cognitives différentes. Il est donc compréhensible que chaque étudiant présente plus ou moins de problèmes en fonction du type d'exercice, de la présentation de la question, etc. [Carmo et al., 2006].

2.2.2 Fixed et Growth Mindset

Les psychologues ont tendance à classer les personnes en deux types de mentalités. L'une sera la mentalité fixe (**fixed mindset**) tandis que l'autre sera la mentalité de croissance (**growth mindset**). Les gens ayant une mentalité fixe auront tendance à croire qu'ils sont nés avec une certaine intelligence et qu'ils ne pourront pas faire grand-chose pour l'influencer. Alors que ceux qui ont une mentalité de croissance auront tendance à penser qu'avec un travail acharné et de la persévérance, l'intelligence ou leur savoir-faire pourra augmenter [Murphy and Thomas, 2008].

Durant l'apprentissage de la programmation, les étudiants de première année se voient confrontés à des obstacles comme des erreurs de codage. Lorsqu'un étudiant avec une mentalité fixe se retrouve dans cette situation, il aura tendance à ressentir un sentiment d'impuissance et souhaitera rapidement abandonner. En revanche, celui avec une mentalité de croissance voudra persévérer pour surmonter cet obstacle [Scott and Ghinea, 2014].

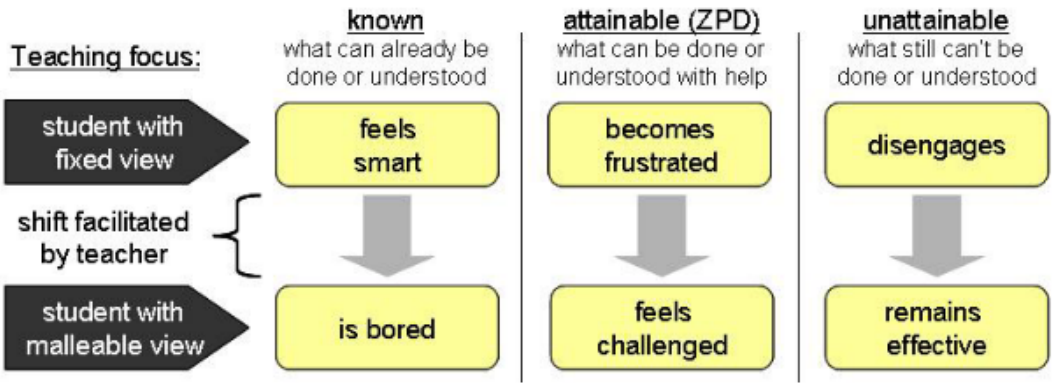


FIGURE 3 – Déplacer un étudiant d'une mentalité à une autre [Murphy and Thomas, 2008].

Toutefois, il a été démontré que cette conception de la mentalité n'est pas immuable et qu'il est possible d'agir positivement sur celle-ci [Murphy and Thomas, 2008].

Sur la figure 3, les émotions sont mises en avant et reflètent ce qui a été exposé dans la section 2.1.3. Il y a plusieurs niveaux de difficulté et dans chacun de ceux-ci, un étudiant réagira de façon différente en fonction de sa mentalité. Le point important de ce graphe concerne les flèches verticales. Il est possible de passer d'une mentalité fixe à une mentalité de croissance. Ce passage est possible grâce à l'enseignant dont la mission est de faire progresser l'étudiant notamment par l'utilisation de feedbacks positifs. Ceci sera exploré dans la section suivante.

2.2.3 Feedbacks positifs

Il est possible d'influencer positivement la mentalité d'un étudiant. Pour ce faire, le **feedback** est un moyen extrêmement important et efficace dans l'évolution d'une mentalité fixe vers une mentalité de croissance. La notion de feedback est quelque chose qui revient très souvent et a montré sa pertinence dans l'apprentissage de la programmation. De manière générale, un feedback négatif aura tendance à pousser à l'abandon l'étudiant doté d'une mentalité fixe. Il est donc très important que le feedback donné à l'étudiant soit positif. En plus d'être positif, celui-ci doit être pertinent afin que l'étudiant puisse avancer dans la résolution de son programme [Cutts et al., 2010]. Les étudiants avec une mentalité de croissance auront beaucoup plus de facilités à être confrontés à une erreur et à essayer de s'améliorer. Il a été remarqué qu'en ajustant le feedback de manière positive aux étudiants avec une mentalité fixe, ceux-ci ont eu tendance à abandonner moins vite. Ces étudiants réagiraient de manière plus positive. Il est donc très intéressant de constater qu'en ajustant la manière de communiquer l'information, un étudiant verra sa détermination et son investissement se modifier [Murphy and Thomas, 2008].

Au-delà des mentalités, il a été démontré que les explications des erreurs fournies par les interpréteurs et compilateurs sont ardues à comprendre pour les

Binary Information	**Low Information**	**High Information**
It works/It doesn't work (can refer to compiles or runs)	I noticed that something I changed altered the output, but I don't know **what** the problem is	I **know** what the problem is, and what I need to do, but I might struggle to actually do it

FIGURE 4 – Les différents types de feedbacks [Kinnunen and Simon, 2010].

débutants. Plus important encore, ces explications peuvent être perçues comme négatives et donc faire abandonner une partie des étudiants. La compréhension de l'erreur est un point clé dans l'apprentissage de l'étudiant [Kinnunen and Simon, 2010].

On distingue trois manières de comprendre et réagir face à une erreur (figure 4) :

- **Binary information** : que cela fonctionne ou pas, je ne comprends pas
- **Low information** : je sais qu'il y a une erreur, mais je ne sais pas où est le problème
- **High information** : je sais quel est le problème et je sais quoi faire, mais je n'y arrive pas.

Il est donc très important d'identifier d'abord dans quel état les étudiants se trouvent afin de les aider efficacement à progresser. Guider l'étudiant dans sa réflexion en fonction de sa compréhension est primordial sinon cela peut conduire à une « assistance » dépourvue de sens pour les élèves [Kinnunen and Simon, 2010].

2.2.4 Motivations

La motivation d'un étudiant à apprendre l'informatique est un facteur clé. De manière générale, un étudiant qui souhaite aboutir se doit d'être motivé et de s'investir. Cela est d'autant plus vrai pour la programmation qui est une discipline avec un aspect pratique important [Jenkins, 2001].

Les différentes formes de motivation ont pu être classifiées en trois grandes catégories qui englobent des motivations plus détaillées [Entwistle, 1998].

- **Extrinsèque**
  - l'apprentissage a un but instrumenté
  - l'étudiant a la satisfaction de la réalisation d'un travail
  - l'étudiant est influencé par des récompenses ou de la pression (p. ex. : pression familiale).

Par exemple, si un étudiant apprend l'informatique dans le but de trouver facilement un travail et de gagner de l'argent (but futur) alors cette motivation est bien extrinsèque.

- **Intrinsèque** :
  - l'apprentissage répond à un objectif personnel
  - l'étudiant démontre un intérêt pour l'informatique
  - l'apprentissage provient d'un choix personnel
  - l'apprentissage va être influencé par des sentiments de confiance et de compétence.

    Par exemple, si un étudiant est passionné par l'informatique depuis son enfance, l'apprentissage de la programmation sera perçu uniquement comme une continuité dans cette passion ; dans ce cas, il s'agit d'une motivation intrinsèque.

- **Réalisations**
  - l'étudiant cherche à entrer en compétition, à se mettre en avant
  - l'étudiant va se concentrer sur les apprentissages qui lui fourniront des acquis personnels
  - l'apprentissage sera influencé par le temps consacré et par l'organisation de l'étudiant
  - l'étudiant verra les travaux comme des challenges personnels

    Par exemple, si un étudiant est motivé pour obtenir les meilleures reconnaissances professionnelles en informatique, il cherchera à avoir les meilleures notes, car elles influenceront sa carrière future ; dans ce cas, il s'agit d'une motivation de réalisations.

Ces trois catégories de motivation conduisent à des manières d'apprendre bien différentes.

Pour une motivation extrinsèque, l'étudiant aura tendance à apprendre de manière non flexible (« par coeur ») et aura des difficultés à transposer ses apprentissages dans d'autres contextes. Une motivation intrinsèque apportera au contraire, un apprentissage plus flexible et permettra une adaptation plus aisée de l'étudiant. Car celui-ci aura cherché à comprendre le contenu en profondeur et de manière plus conceptuelle. Finalement, une motivation au niveau des réalisations va mener à une approche stratégique et polyvalente de l'apprentissage [Entwistle, 1998].

En Bref ! : Facteurs psychologiques

Cette section met en avant quatre facteurs psychologiques. Bien que difficile à définir, le profil d'apprentissage permet de renforcer l'idée que chaque individu a sa propre façon d'apprendre. Les mentalités "fixed" et "growth mindset" permettent d'identifier comment une personne va réagir par rapport à un exercice. Avoir un certain type de mentalité n'est pas immuable. Les feedbacks ont la capacité à faire basculer une mentalité fixe vers une mentalité de croissance. Enfin, l'apprentissage est influencé par la motivation de l'étudiant. En fonction de sa motivation, celui-ci aura plus ou moins de difficultés dans son apprentissage.

2.3 Difficultés et Misconceptions

Les difficultés et les misconceptions sont deux éléments clés qui permettent de définir les problèmes que les étudiants peuvent rencontrer dans leur compréhension de la matière et principalement lors de son application à la résolution d'un exercice.

Zingaro et d'autres proposent une définition de ce qu'est une "misconception" :

> A student conception describes a belief, theory or explanation previously developed to explain some behavior observed in the world. When these beliefs are in conflict with accepted scientific theories, they become misconceptions. A difficulty refers to an observable error committed by students [Zingaro et al., 2018].

Les difficultés et les misconceptions semblent universelles. En effet, ces notions ne sont pas propres au monde de l'informatique.

2.3.1 Difficultés

Nombreuses sont les difficultés rencontrées par les étudiants en programmation. Celles-ci peuvent être liées à la compréhension de concepts tels que les notions de boucles et de conditions ou encore aux concepts de procédures et de fonctions. D'autres embûches sont également à prendre en compte et à considérer lors de l'élaboration d'un exercice. Parmi celles-ci, on peut citer la gestion de l'environnement de programmation qui peut être déroutante de prime abord, la gestion des fichiers en entrée/sortie ou encore la gestion des listes [Robins et al., 2006].

L'apprentissage est une accumulation de connaissances et de nouveaux concepts. Ceux-ci s'additionnent et s'ils ne sont pas parfaitement intégrés, les difficultés des étudiants augmentent significativement [Cherenkova et al., 2014]. Il faut visualiser cette accumulation comme une pyramide où la base est le concept élémentaire et les étages suivants sont des concepts de complexité croissante. Chaque concept repose sur d'anciens concepts appris préalablement.

Par exemple, le concept de liste est assez rapidement lié au concept de boucle, car ils sont dans la réalité bien souvent utilisés ensemble. Une difficulté dans la compréhension des boucles rendra encore plus difficile la compréhension des listes pour un étudiant.

Il est intéressant de constater que bon nombre d'étudiants arrivent heureusement à surpasser leurs difficultés. Cela se traduit par la diminution au fil du temps du nombre de questions posées. Par exemple, les questions sur les boucles et les conditions vont diminuer, mais pas celles sur la visibilité, le scope, les références, etc. [Cherenkova et al., 2014].

2.3.2 Misconceptions

Les **misconceptions** font donc référence à des compréhensions fausses ou inadéquates et deviennent seulement visibles lors de leur mise en place dans un contexte informatique et pratique [Qian and Lehman, 2017]. L'informatique est une science bien définie qui catégorise l'ensemble de ses connaissances. En définissant celles-ci, les misconceptions se rapporteront automatiquement à une catégorie bien spécifique.

L'informatique peut être séparée en trois catégories [Qian and Lehman, 2017] de connaissances(figure 5) :

1. **Syntaxique** : reprend les connaissances nécessaires à l'expression d'une idée d'une manière programmatrice ;
2. **Conceptuelle** : reprend tous les concepts de base de la programmation, les concepts clés (p. ex. : variables) ;
3. **Stratégique** : reprend les connaissances nécessaires pour pouvoir concevoir une solution de A à Z, en appliquant la bonne méthodologie, adaptée à la situation.

Dans la figure 5, on remarque, par exemple, que les erreurs associées à une mauvaise utilisation de l'opérateur de comparaison (==) font partie des erreurs syntaxiques. Afin d'aider ces étudiants, l'utilisation d'un environnement de développement relativement évolué permet de signaler facilement ce genre de mauvaise compréhension. Ce dernier pourra souligner ces erreurs et afficher le message explicatif associé.

En Bref ! : Difficultés et Misconceptions

Cette section met en évidence deux concepts clés qui permettent d'évaluer les problèmes rencontrés par des étudiants. Ces problèmes ont été mis en lumière à travers le détail de leurs difficultés. Une difficulté fait référence à une compréhension partielle ou erronée d'un concept. Ensuite, les misconceptions ont été définies comme de mauvaises compréhensions de la part des étudiants et qui ne peuvent être révélées que par la réalisation de travaux pratiques.

Knowledge	Exemplars of Difficulties	Major Related Factors	Potential Strategies and Tools
Syntactic	Syntax errors such as • Mismatched parentheses *while ((a > b) && (a>c) { … }* • Incorrect use of the comparison operator *if (a = true) { … }*	• Task complexity can increase cognitive load and cause novices to forget basic syntax. • Programming environments may not support highlighting errors.	• Advanced editors or graphical programming environments can highlight or prevent syntax errors, reduce cognitive load, and help students with syntactical difficulties.
Conceptual	Misunderstanding of programming constructs or machine operation such as • Variables and assignment statements • Conditional expressions • Loops • Classes and objects • Sequential code execution	• Knowledge of natural language may interfere with learning of English-like statements. • Prior math knowledge may interfere with understanding of computer expressions. • Students' mental models of computer operation may be flawed. • Teachers' instruction and limited knowledge may contribute to students' misconceptions. • Some computing environments create conceptual difficulty by using the same symbol (+) for multiple functions (addition, concatenation).	• Well-chosen program examples can help students build accurate understanding of programming and improve knowledge transfer. • Visualization tools such as Python Tutor can help students to visualize code execution step by step and build correct mental models. • Natural-language-like programming languages such as MOOSE may reduce natural-language errors. • According to conceptual change theories, analyzing students' existing knowledge elements may help instructors better understand students' misconceptions to help them refine and reorganize their knowledge to be more complete and utilitarian. • A concept inventory can help instructors evaluate students' understanding of programming concepts. • Peer instruction can help students improve their understanding of concepts in programming.
Strategic	Difficulties applying programming to solve problems such as • Failing to choose an appropriate loop construct in a specific context • Failing to test and debug programs	• Novices often have fragmentary knowledge with an inadequate set of patterns and strategies for problem solving.	• Asking students to read and trace example programs can help students develop strategic knowledge. • Explicitly teaching debugging strategies and using enhanced debugging tools (e.g., providing detailed error messages) may improve students' debugging skills.

FIGURE 5 – Exemples de difficultés et de misconceptions [Qian and Lehman, 2017].

2.4 Les IDEs

Le choix d'un Integrated Development Environment (IDE) ou Environnement de Développement Intégré (EDI) est crucial dans un cours où l'on enseigne la programmation. Ce choix doit être motivé par différents aspects : la prise en main, le bruit qu'il peut générer en ce qui concerne son utilisation pour un débutant, mais aussi son intérêt dans le monde de l'industrie. En effet, certains IDE sont conçus dans un objectif bien précis, soit pédagogique soit professionnel. Dans ces deux situations, les utilisateurs visés sont très différents et le produit qui en résulte l'est également. Il est également intéressant de prendre en compte que certains IDE peuvent être payants [Mason et al., 2015].

Le choix d'un IDE repose sur de nombreux critères. Néanmoins, cinq critères reviennent de manière récurrente lorsqu'il s'agit de choisir un IDE ayant pour but l'apprentissage de la programmation. Ces critères sont les suivants :

- pédagogique : la capacité de l'IDE à aider l'étudiant dans l'apprentissage de la programmation
- debugger/repères visuels : le fait que l'IDE soit clair
- facile à utiliser : le fait que la prise en main de l'IDE puisse se faire rapidement
- disponibilité/coût en apprentissage pour l'étudiant : ressemble fort aux points précédents, le but est que l'IDE ne surcharge pas l'étudiant
- motivation des étudiants : si les étudiants ont envie d'apprendre sur cet IDE (p. ex. : les étudiants peuvent formuler leur avis sur les IDE et

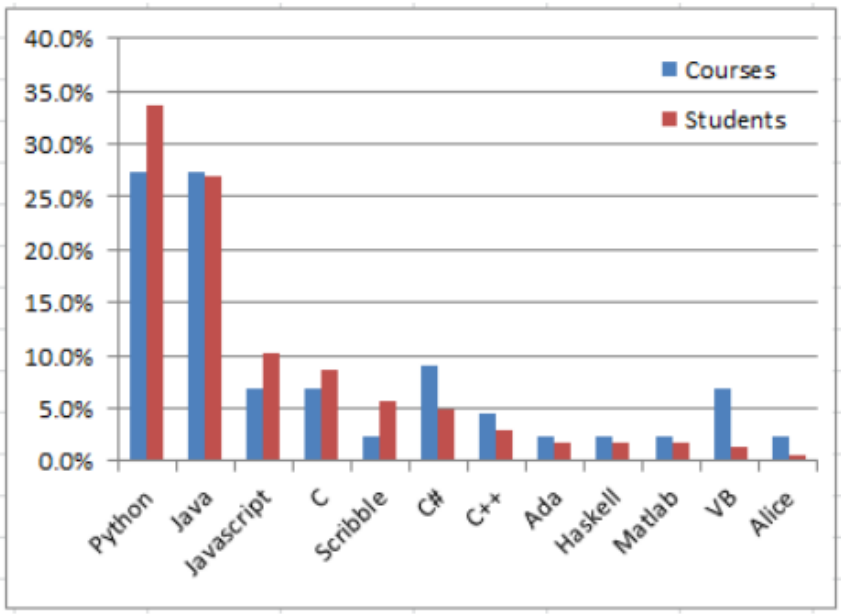


FIGURE 6 – Pourcentage d'étudiants apprenant un certain langage de programmation et pourcentage de cours utilisant un certain langage de programmation [Mason and Cooper, 2014].

l'année suivante, celui-ci pourrait être pris en compte)

L'utilisation d'un IDE s'accompagne de difficultés. En effet, il faut être particulièrement vigilant à la charge cognitive que va induire l'utilisation de tel ou tel IDE et prendre en compte la limite de la mémoire de travail (sept éléments en simultané) (voir section 2.1.2). Pour contourner cette limitation, l'utilisateur créera dès lors du sens dans son environnement ou le concepteur de l'IDE veillera à ne pas multiplier les interactions possibles.

Afin d'être plus représentatif, des exemples d'IDE et de leurs avantages et inconvénients seront détaillés dans la section 2.6.

2.5 Les différents langages

Chaque langage de programmation a ses propres spécificités et son utilisation dans le monde de l'industrie peut varier. Au fur et à mesure des années, Python et Java sont devenus des « standards » pour l'apprentissage de la programmation (figure 6). Le langage pour apprendre la programmation est très souvent sélectionné en fonction de deux critères. Le premier critère étant sa présence dans l'industrie et le deuxième étant sa « capacité à être pédagogique ». Évidemment il existe bien d'autres critères comme la facilité d'installation, la présence d'une Graphical User Interface (GUI) ou encore pouvoir faire de la Programmation Orientée Objet (POO), etc. [Mason and Cooper, 2014]. Pour ne citer que les plus utilisés à l'heure actuelle, Python présente l'avantage d'avoir une syntaxe très « légère » et peu de contraintes. Ceci est un grand avantage pour la gestion de la charge cognitive. L'utilisation d'un langage de programmation plus simple d'utilisation permet aux étudiants de se concentrer

davantage sur l'exercice en lui-même. C'est d'ailleurs pour cette raison que Java a perdu du terrain par rapport à Python, en raison de ses règles de syntaxe beaucoup plus lourdes. Toutefois, Java reste toujours le langage numéro 1 dans l'apprentissage de la programmation orientée objet [Mason and Cooper, 2014]. Au niveau de la figure 6, JavaScript se retrouve en troisième position dans l'apprentissage de la programmation. Celui-ci a pour avantage d'être facile à installer et les résultats produits par un code dans ce langage sont très visuels. En effet, ce langage se couple très souvent avec HTML ainsi que CSS et permet d'obtenir un ensemble très interactif.

Le langage de programmation ne doit pas être une fin en soi, mais un moyen pour l'étudiant à apprendre la programmation de façon plus aisée. C'est pour cette raison que Python commence à devenir un standard dans les cours de première année [Berglund and Lister, 2010].

2.6 Solutions existantes

Les étudiants disposent de plusieurs solutions pour répondre au mieux à leurs besoins.

Une solution connue et souvent utilisée est *Thonny*, un IDE donnant des outils afin de faciliter la programmation des débutants. La volonté de *Thonny* est d'améliorer l'accessibilité d'un IDE en utilisant, notamment, des moyens visuels qui mettent en avant l'exécution du code, les scopes, les erreurs de règles de syntaxe, etc. La grande force de *Thonny* est son débogueur. L'interface utilisateur de *Thonny* est adaptée aux débutants (figure 7). Les étudiants, qui ont décidé d'installer *Thonny* sur leur propre ordinateur, ont pu utiliser ses fonctionnalités sans autre aide [Annamaa, 2015]. *Thonny* utilise exclusivement le langage Python. La faiblesse de *Thonny* est qu'il n'est pas du tout adapté à la construction de gros programmes informatiques impliquant un grand nombre de lignes de code ou demandant une architecture découpée en plusieurs fichiers.

Dans un même ordre d'idées, *UUhistle* est un programme permettant de visualiser l'exécution du code en Python également. Ce programme présente une interface graphique montrant l'exécution du code pas à pas. Le programme se veut très accessible pour les étudiants et ainsi leur permettre de facilement manipuler les éléments de programmation (figure 8) [Sorva and Sirkiä, 2010]. De même que *Thonny*, *UUhistle* est très pratique tout au début, mais celui-ci perd rapidement de son intérêt, car il n'est pas du tout adapté aux programmes informatiques de plus grande ampleur.

BlueJ est un autre exemple de logiciel qui permet l'apprentissage de la programmation orientée objet à travers l'utilisation du langage Java [Kölling et al., 2003]. Au contraire des deux solutions détaillées plus haut, *BlueJ* ne met pas en avant l'exécution du programme, mais permet de visualiser les différents concepts de la programmation orientée objet grâce à de l'UML (figure 9) [Sorva and Sirkiä, 2010].

BlueJ est conçu comme un environnement d'apprentissage, pour une introduction et n'est pas un logiciel destiné à être utilisé dans l'industrie [Sorva and

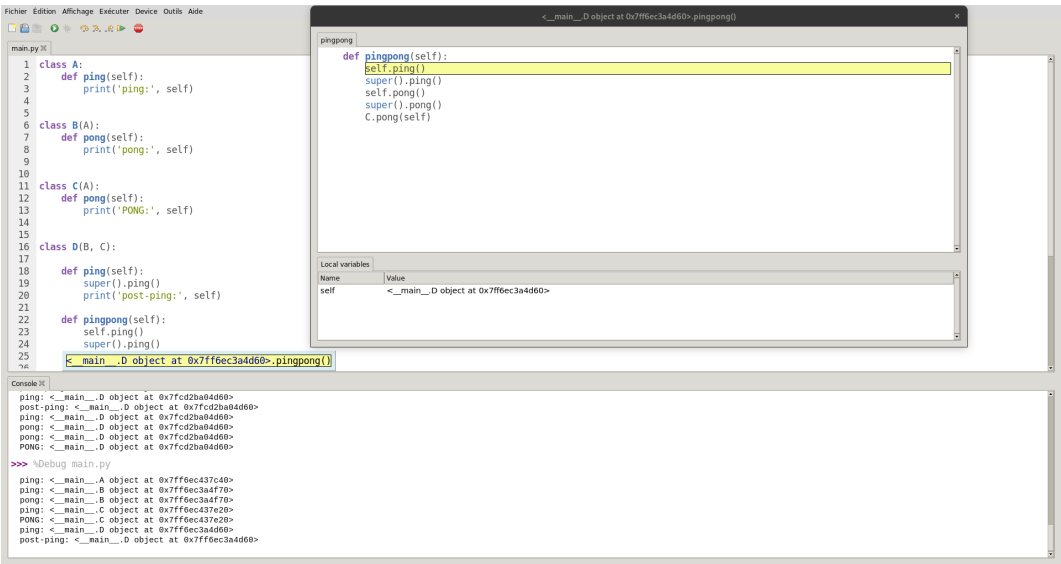


FIGURE 7 – Exemple d'utilisation de *Thonny*.

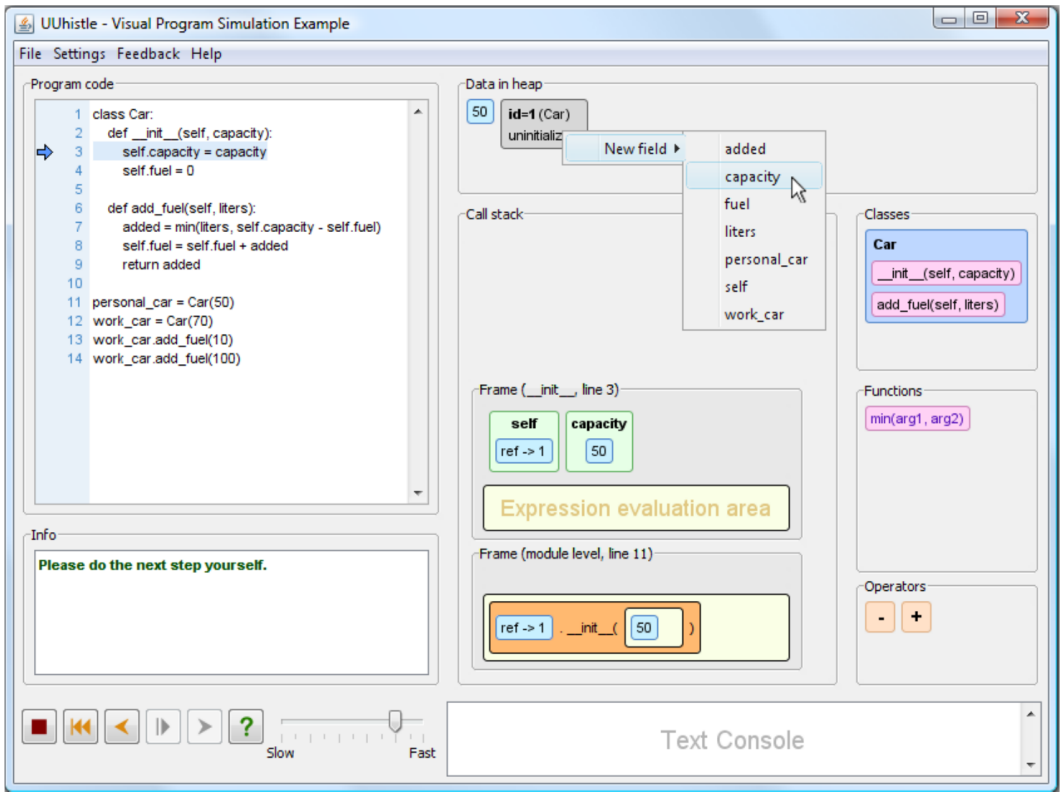


FIGURE 8 – Exemple d'utilisation de *UUhistle* [Sorva and Sirkiä, 2010].

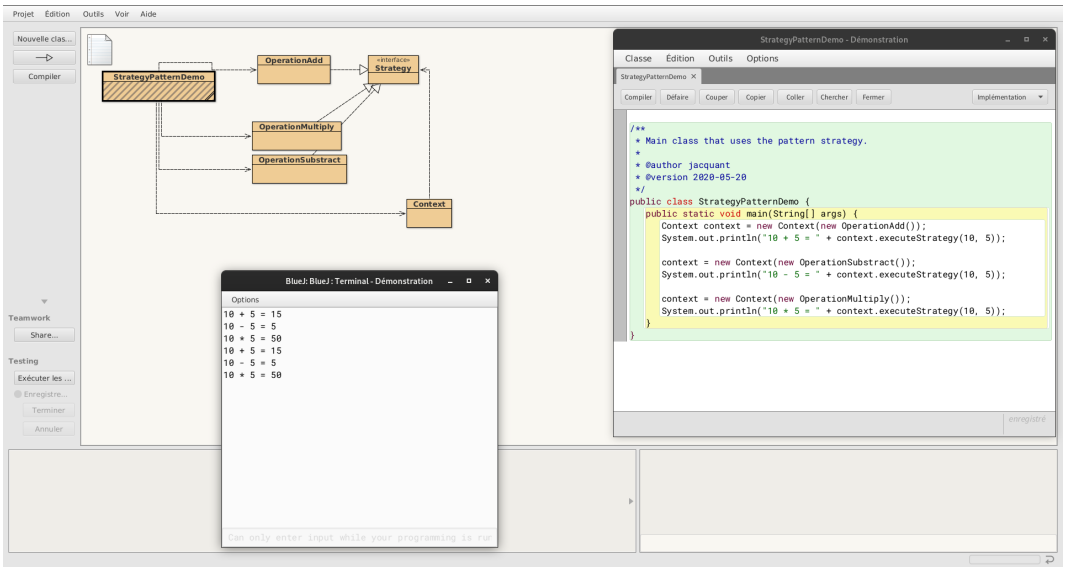


FIGURE 9 – Exemple d'utilisation de *BlueJ*.

Sirkiä, 2010].

Outre ceux-ci, il existe d'autres logiciels, mais aucun n'a pour vocation d'être utilisé dans l'industrie. Cela peut provoquer une certaine détresse pour l'étudiant lors du passage d'un logiciel pédagogique à un logiciel professionnel.

2.7 Chatbot

Des chatbots existent dans de nombreux domaines. C'est une aide pour les utilisateurs. Ils facilitent le travail et l'interaction avec le domaine dans lequel ils évoluent en utilisant le langage naturel [Abu Shawar and Atwell, 2007].

Le chatbot est très souvent utilisé dans un cadre pédagogique et possède sa propre manière de communiquer de l'information. En effet, il s'adapte en fonction de la personne qui est en train de lui parler [Benotti et al., 2014]. Sur base des interactions écrites échangées, le chatbot utilisera des patterns dont l'intérêt est de trouver la bonne motivation pour les étudiants afin de leur enseigner les concepts de base. Un chatbot se base sur ce que la personne lui écrit, il utilise des patterns.

Selon *Benotti*, l'impact des chatbots est positif pour les étudiants dans leur apprentissage de la programmation. Les étudiants ont tendance à trouver qu'un chatbot est relativement intéressant, pratique et donne envie d'en apprendre plus. Cela est notamment expliqué par le fait qu'un chatbot procure à l'étudiant un retour d'information immédiat. De plus, l'information est donnée de manière structurée afin de ne pas perdre l'étudiant [Benotti et al., 2014].

Néanmoins, même si l'utilisation d'un chatbot dans le domaine de l'apprentissage peut servir d'amplificateur, il faut constater qu'il ne peut remplacer un professeur. En effet, le chatbot reste limité dans ses interactions en fonction de sa base de connaissances et celui-ci n'a pas toujours la capacité de saisir le contexte derrière une question [Abu Shawar and Atwell, 2007].

Résumé du chapitre ► État de l'Art

Dans ce chapitre, l'attention a été placée sur les informations utiles pour comprendre l'apprentissage d'un étudiant et comment celui-ci peut être influencé. Cet apprentissage est donc impacté par des facteurs cognitifs (en lien avec ce qui est mis en place pour l'apprentissage) et des facteurs psychologiques (en lien direct avec la personnalité de l'étudiant).
La solution s'intéresse également aux erreurs produites par les étudiants. Le chatbot s'inscrira dans une utilisation pratique (réalisation de TP). Celui-ci permettra donc de détecter les misconceptions des élèves et permettre au corps enseignant de se repositionner par rapport à celles-ci.
Ce chapitre a également mis en évidence, autant dans un cadre pédagogique que professionnel, les utilisations d'IDE, les choix des langages de programmation et les solutions préexistantes.

3 Conception

Le travail a consisté en la réalisation d'une plateforme d'aide à la programmation. Les différentes étapes depuis la conception jusqu'à la mise en service vont être détaillées.

3.1 Observations

Après analyse des solutions existantes, passées en revue dans le chapitre précédent, l'observation des moyens utilisés par les étudiants a permis d'évaluer le degré d'adaptation des outils utilisés pour leur apprentissage de la programmation.

3.1.1 Public observé

Durant la réalisation du mémoire, le cours *[INFOB132] Projet de programmation* était également donné. Lors de ce cours, des étudiants de bloc 1 en fin d'apprentissage de la programmation ont été rencontrés. Ils devaient mettre en pratique ce qu'ils avaient étudié durant le premier quadrimestre. L'approche avec ce public s'est déroulée de deux manières.

1. Rencontre des étudiants de manière individuelle et collective
2. Utilisation d'un « HelpDesk » permettant aux étudiants de venir poser leurs questions sur une plateforme en ligne.

Concernant les rencontres individuelles, plusieurs questions ouvertes ont été posées afin de cibler les difficultés, les outils utilisés, les moyens de résolution des erreurs, etc. Le but de ces questions était de déterminer les différents profils d'étudiants et de bien comprendre les difficultés rencontrées, mais également les points positifs durant l'apprentissage de la programmation.

Toutes les questions du tableau 1 ont été posées juste avant la fin du premier quadrimestre à plusieurs étudiants en informatique ou en ingénieur de gestion. Les résultats de ces observations seront explicités dans la section 3.1.2.

L'approche « HelpDesk » avait pour but de récolter un grand nombre de données. Les étudiants ont été amenés à poser leurs questions au HelpDesk lorsqu'ils étaient confrontés à un problème dans leur projet. L'objectif était double : d'abord la récolte des questions posées et ainsi dégager des tendances ; ensuite le paramétrage des réponses prédéfinies à des questions et l'observation des réactions de l'étudiant vis-à-vis de ces réponses automatiques. Le fonctionnement du HelpDesk était très simple, les étudiants posaient leurs questions sur la plateforme (figure 10). En fonction de la question, si celle-ci était « simple », une réponse automatique était donnée. Si la question était trop complexe, une véritable personne intervenait et répondait à la question. Dans tous les cas (réponse automatique ou non), la réponse comprenait un rappel théorique de la partie qui posait problème. Par exemple, s'il y avait un

Questions	**Réponses**
Quelle est ton expérience avec l'informatique ?	
Avais-tu déjà programmé avant d'être à l'université ?	
Si oui qu'as-tu déjà réalisé ? Si non, pourquoi ?	
Comment trouves-tu le cours d'introduction à la programmation ?	
As-tu bien saisi tous les concepts qui ont été vus en cours ?	
Quand tu bloques lors de la réalisation d'un exercice Comment réagis-tu ?	
Est-ce que la barrière de la langue est un problème ?	
Quelle est ton approche avec la documentation en ligne ?	
À quel moment te sens-tu capable de coder seul ? et sans être accompagné ?	
As-tu eu des difficultés avec les outils qui servent à programmer ?	
Comment améliorais-tu la notion d'apprentissage de la programmation ?	

TABLE 1 – Questions posées aux étudiants lors d'interviews

problème avec les boucles, un rappel théorique des boucles était donné. En plus de cela, les « erreurs classiques » étaient présentées et un lien était donné vers une documentation [2] non officielle, mais très réputée en français. Toutefois, le HelpDesk n'a pas permis de mettre le doigt sur des misconceptions. À l'inverse, les erreurs de compréhension ont pu être clairement identifiées. Le niveau de connaissance en informatique des étudiants a rendu plus laborieux leur capacité à écrire de façon compréhensible leurs difficultés face au problème rencontré.

En Bref ! : Public observé

Pour réussir à proposer un chatbot le plus adapté aux étudiants, nous avons procédé en deux grandes étapes :

1. Comprendre et identifier les différents profils des étudiants afin d'observer les difficultés.
2. Tester en situation réelle un chatbot (appelé HelpDesk) pour distinguer les différentes utilisations et obtenir les questions posées.

3.1.2 Analyse des résultats

Les rencontres avec les étudiants ont permis de mieux comprendre leurs besoins et distinguer les différents types de profils. Certains étudiants sont plus à l'aise avec un langage de programmation tandis que d'autres ont plus de difficultés avec les concepts de base et posent des questions très abstraites. N'ayant

2. https://python.doctor

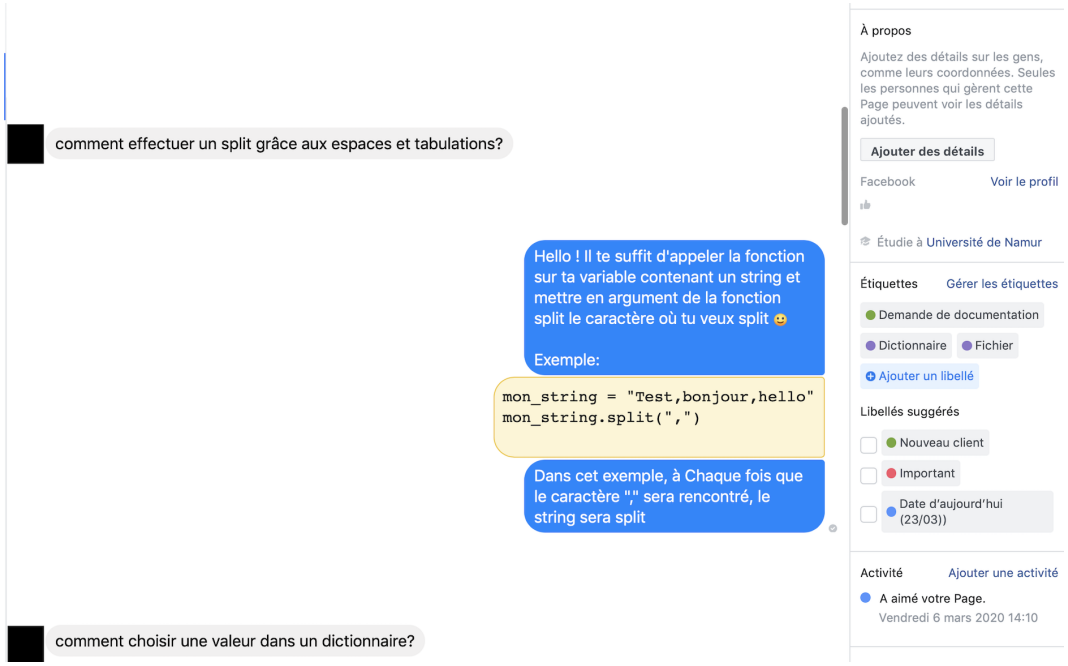


FIGURE 10 – HelpDesk mis en place.

pas compris les concepts se cachant derrière l'exercice, les étudiants posent une question du type *« Pourquoi mon dictionnaire n'a-t-il pas ajouté la valeur ? »*, *« Pourquoi j'obtiens ce résultat ? »*, etc. D'un autre côté, ceux qui maîtrisent la matière ont tendance à poser des questions plus précises *« Comment utiliser split ? »*, *« Comment supprimer un élément de mon dictionnaire ? »*, etc. Il est évident qu'il s'agit d'un résumé simpliste, mais cela permet de se rendre compte des types de questions que ces groupes d'étudiants sont amenés à poser. Grâce au HelpDesk, il a été possible d'affiner les types de questions posées pour concevoir un chatbot. Il n'a malheureusement pas été possible d'explorer plus en profondeur la piste des rencontres des étudiants à la suite de la crise sanitaire du coronavirus.

Un point important à soulever durant les échanges concerne la documentation. La majorité des étudiants préfèrent les exemples et trouvent qu'il n'est pas toujours simple de lire une documentation. La présentation de l'information semble jouer énormément sur la compréhension de l'étudiant et même sur sa motivation à essayer de comprendre.

Le HelpDesk a permis d'obtenir beaucoup d'informations. Les structures de données et la manipulation de fichiers ont été des sujets de question très récurrents. Ce n'est pas étonnant, car ces sujets ont tendance à être plus abstraits que la notion de « condition » par exemple. Lorsque l'étudiant recevait sa réponse, obtenir une réponse en français suffisait à le rassurer et le mettait dans de bonnes dispositions.

Au fur et à mesure de l'arrivée des questions, un regroupement a été réalisé afin de pouvoir identifier les questions récurrentes.

La première grande tendance concerne les questions sur le contexte. La clarté de l'énoncé, les énoncés ambigus, etc. ont fait l'objet d'un grand nombre

de questions. C'est très intéressant de voir que l'énoncé en lui-même peut être un obstacle et freine l'étudiant dans la réalisation d'une problématique.

La deuxième tendance, déjà brièvement évoquée, concerne les questions sur la matière en elle-même. Beaucoup de questions impliquaient des « erreurs classiques » comme un problème de structure dans les dictionnaires, d'itération dans les boucles, dans l'ouverture d'un fichier, etc.

La troisième tendance concerne les questions très précises sur l'exercice et certains étudiants souhaitaient de façon claire la réponse directe au problème. Plusieurs raisons peuvent permettre de comprendre cette demande. Par exemple, il peut s'agir d'un souci de compréhension ou d'un manque de connaissances de la programmation. Il peut également s'agir d'un manque d'intérêt. Il n'est pas toujours évident de comprendre ce qu'il se cache derrière une question d'un étudiant. Le constat qui a été tiré avec ce genre de questions était qu'il n'y avait aucune démarche constructive de la part des étudiants pour essayer de s'améliorer. Avec du recul, la troisième catégorie est sans aucun doute la plus complexe à traiter.

Lorsque l'on se concentre sur le feedback envoyé aux étudiants, le fait de rester en français et d'être plus explicite qu'une documentation leur a permis de continuer à avancer. Il est très laborieux de savoir si les feedbacks donnés avec le HelpDesk se sont avérés être plus pertinents avec des étudiants ayant une mentalité fixe, mais le fait de voir des étudiants revenir plusieurs fois pour des questions différentes a laissé entrevoir des points positifs. Des étudiants ayant de réelles difficultés ont réussi à comprendre leurs erreurs et sont revenus à chaque fois qu'ils en rencontraient de nouvelles. Le fait de dialoguer en donnant des feedbacks qui ont été orientés « positifs » n'a, en tout cas, pas démotivé les étudiants. Un autre avantage à tirer du HelpDesk est sa simplicité. L'étudiant n'avait qu'à poser sa question et attendre une réponse. Aucune autre information n'était présente afin d'éviter de complexifier la compréhension du problème.

En Bref ! : Analyse des résultats

Grâce aux rencontres et au HelpDesk, il a été possible de déterminer un ensemble de questions pouvant être posées par les étudiants. Ces questions posées peuvent être classées en trois grandes catégories. La manière dont sera adressé le feedback est notamment basée sur l'expérience des étudiants à propos du HelpDesk.

3.1.3 Utilisation potentielle des résultats

Grâce au HelpDesk et aux rencontres effectuées, il a fallu développer une solution pertinente quant à l'apprentissage de la programmation. Le chatbot est un vecteur de pédagogie qui a semblé pertinent lors des interviews. L'HelpDesk a fait ses preuves durant le confinement. Les étudiants sont venus poser leurs

questions et l'ont utilisé comme un assistant virtuel. Le fait de pouvoir dialoguer avec quelqu'un facilite l'interaction et la compréhension de certains concepts. Les questions qui ont été posées sur le HelpDesk ont permis d'enrichir la base de connaissances du chatbot. Chaque étudiant ayant sa propre façon de parler, il était nécessaire d'encoder une même question posée plusieurs fois de différentes façons pour que le chatbot puisse comprendre la question.

La façon dont l'information était présentée avec le HelpDesk s'est avérée efficace. Généralement après une question, un bref rappel théorique était réalisé puis un exemple était donné. Cela a permis de confirmer l'importance du mode de communication de l'information, et notamment l'intérêt de fournir un exemple. Au niveau du fond et de la forme, le chatbot gardera donc la même structure. Donner l'information en français paraît être judicieux et évite de complexifier la compréhension de l'étudiant. Dialoguer en français à travers le HelpDesk a enlevé cette barrière de la langue. Il paraît dès lors approprié de la conserver pour concevoir le futur chatbot.

En Bref ! : Utilisation potentielle des résultats

Ce sont les questions posées par les étudiants qui ont nourri la première version du chatbot. Un feedback correct adressé aux étudiants a montré son importance et la forme utilisée, appréciée par eux, a été conservée puis adaptée pour le développement du chatbot.

En Bref ! : Observations

À travers cette section, nous avons pu discuter du public observé ainsi que des résultats en rapport avec ces observations. Le public ayant été les étudiants du premier bloc. Les informations récoltées sont notamment les questions posées par un étudiant, mais également les difficultés rencontrées par celui-ci. Les résultats obtenus permettant de nourrir la solution finale du chatbot.

3.2 Définition des bases du travail

Cette section aborde les choix effectués concernant le langage de programmation ainsi que l'Environnement de Développement Intégré (EDI).

3.2.1 Choix du langage

Sur base de l'état de l'art et du cours *[INFOB131] Introduction à la programmation*, le choix du langage s'est porté sur Python. Il est évident que changer de langage de programmation aurait été quelque chose de fastidieux

pour le titulaire du cours et surtout cela n'était clairement pas nécessaire. La pertinence du langage Python a été démontrée dans l'état de l'art et durant les interviews. Ce langage de programmation n'a pas été identifié comme étant un frein par les étudiants ; au contraire, celui-ci possède de nombreux avantages pédagogiques.

3.2.2 Choix de l'IDE

Aujourd'hui, la faculté d'informatique de l'UNamur utilise *Canopy* comme IDE pour ses cours d'introduction à la programmation. Ce logiciel a l'avantage d'être suffisamment ludique pour être utilisé par les primo-arrivants en informatique, mais également d'inclure la gestion et la création de **notebooks**. L'utilisation de ces notebooks est mise en avant dans les cours et les étudiants apprennent donc dans ce type d'environnement bien que sa conception s'éloigne d'une solution professionnelle. L'objectif premier d'un notebook est de conserver une trace des exécutions et de présenter le raisonnement derrière un code. Cependant, on remarque que certains étudiants continuent d'utiliser ces solutions alors que le cadre de réalisation ne s'y prête pas (p. ex. pour d'autres projets informatiques qu'ils sont amenés à réaliser dans la suite de leur cursus universitaire). L'observation des étudiants a permis de remarquer une grande disparité entre les IDE utilisés par les étudiants. Dans les faits, les étudiants qui utilisent *Canopy* ne sont pas majoritaires.

Précédemment, Visual Studio Code (VSCode) a été présenté comme une solution qui se positionne entre l'IDE pédagogique et l'IDE professionnel. Le choix d'un tel IDE permet d'envisager plus sereinement une transition dans les IDE utilisés par les étudiants en comparaison à une solution comme *Canopy*. De manière naturelle, beaucoup d'étudiants choisissent VSCode comme IDE bien qu'il ne soit pas mis en avant dans le cadre des cours d'introduction.

Visual Studio Code permet l'installation d'extensions au travers d'un "Market" de manière facile (One-Click Install). L'intégration d'un chatbot dans un IDE semble adéquate et d'autant plus pour VSCode grâce à ses possibilités de contrôle de la mise en forme. C'est en prenant en compte tous ces critères, que VSCode a été sélectionné pour supporter notre solution de chatbot.

En Bref ! : Définition des bases du travail

Sur base de la revue de littérature ainsi que des observations réalisées, le langage de programmation Python ainsi que sur l'IDE VSCode ont été sélectionnés.

3.3 Solution proposée

Dans cette section, l'infrastructure du projet est détaillée afin de rencontrer les objectifs du travail. La solution se décompose de façon suivante (figure 11) :

1. un chatbot, pièce principale de cette solution.
2. une interface pédagogique pour les professeurs et assistants.
3. un serveur regroupant plusieurs sous-composants permettant de fournir les ressources nécessaires au bon fonctionnement du chatbot et de l'interface pédagogique.

Chaque composant possède sa propre particularité et sert une fonction bien définie dans la réalisation de la solution par rapport aux objectifs posés. Sur base des solutions existantes et après s'être entretenu avec le public cible, développer un nouveau moyen pédagogique d'apprentissage à la programmation s'est imposé comme une évidence.

Pourtant, il est très difficile de trouver un juste équilibre entre les solutions proposées par le monde professionnel et le monde pédagogique. En effet une solution à vocation uniquement pédagogique (p. ex. *Thonny*) devient contre-productive dans le monde professionnel. À l'inverse, une solution professionnelle (telle que *Pycharm*) surcharge cognitivement (voir section 2.1.2) un étudiant avec du contenu superflu pour son apprentissage de la programmation. La solution proposée doit donc trouver un équilibre entre les deux mondes, permettre l'apprentissage d'un premier langage de programmation et même encourager l'étudiant à transiter vers les outils professionnels.

La solution proposée met également l'accent sur les besoins d'interaction des étudiants. Lors de TP, par exemple, ils cherchent à obtenir des réponses claires de la part de leurs assistants sur des points spécifiques. La même qualité de réponse sera, par ailleurs, rarement obtenue sur internet, en raison de la langue employée et la mauvaise interprétation du contenu de la console. Le chatbot répond à cette nécessité. Il simule l'interaction avec un enseignant lorsque des questions basiques de programmation sont posées et stimule l'étudiant afin que celui-ci recherche la réponse de lui-même.

Par ailleurs, un autre but poursuivi par la solution proposée est de soulager les assistants de la surcharge de questions basiques. En effet, si celles-ci sont traitées par le chatbot, un gain de temps substantiel sera dégagé afin de pouvoir se concentrer sur des questions plus complexes. Enfin le chatbot a pour intention d'être complémentaire avec les séances de TP.

3.3.1 Chatbot

Pour créer un agent conversationnel, la librairie *ChatterBot*[3]a été utilisée, car elle met en place un nombre d'outils utiles pour la création du chatbot souhaité. Pour fonctionner, le chatbot a besoin d'être entraîné avec un certain nombre de questions/réponses afin que par la suite, il puisse comprendre la question posée par l'utilisateur, peu importe le phrasé. Dans ce cas-ci puisqu'il s'agit d'un agent conversationnel pour l'apprentissage de la programmation, il est primordial que cet entraînement soit réalisé par un spécialiste du langage

3. https://chatterbot.readthedocs.io/en/stable/

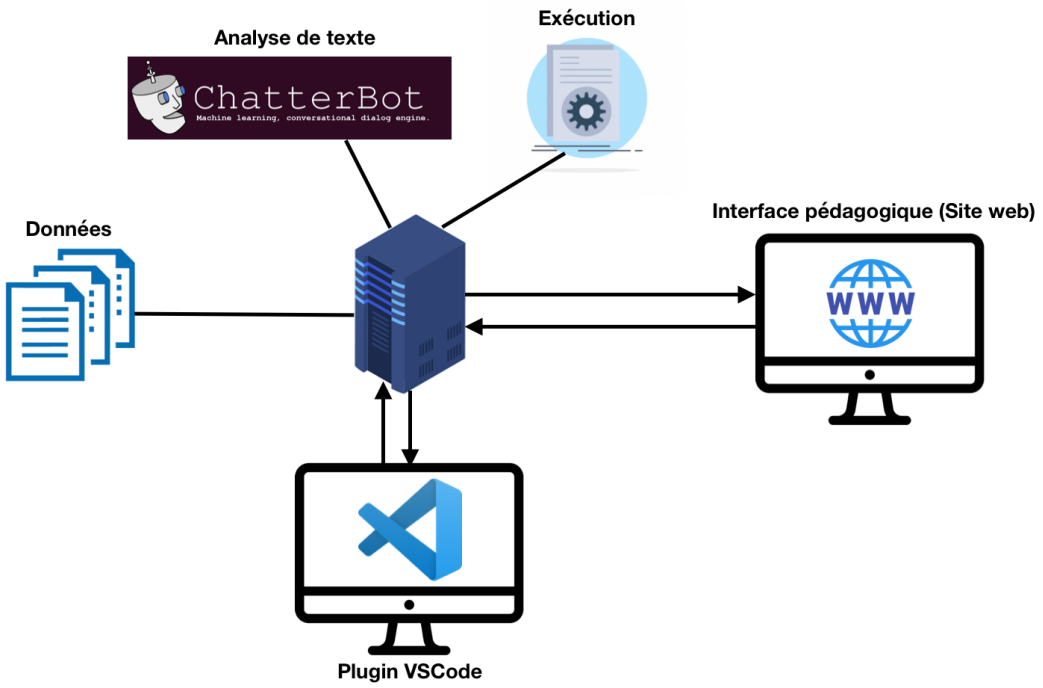


FIGURE 11 – Infrastructure proposée pour répondre aux objectifs du mémoire.

Python afin d'apporter des réponses cohérentes à des questions classiques posées par les étudiants. Le bot va donc enregistrer dans sa base de données un ensemble de questions/réponses ainsi qu'éventuellement des fils de discussions si cela s'y prête (figure 12).

Une fois le chatbot entraîné, il est possible de lui poser des questions en lien avec l'entraînement qu'il a reçu. Dès que l'étudiant a posé sa question, un processus est mis en place afin de trouver le meilleur "matching" possible avec une question préenregistrée dans le bot (figure 13) .

Pour déterminer si la question posée correspond à un élément introduit en BD, la *distance de Levenshtein* est utilisée. Celle-ci permet de donner la différence entre deux chaînes de caractères de manière mathématique. Lorsque l'on prend deux chaînes de caractères et qu'on y applique *l'algorithme de Levenshtein*, on obtient ainsi le coût minimal pour que la première chaîne de caractères devienne identique à la seconde (figure 14). Il est important de noter que lors de la comparaison de chaînes de caractères, elles sont au préalable nettoyées de tous les petits mots « inutiles » (adverbes, prépositions, déterminants, etc.). Pour ce faire, la librairie Natural Language ToolKit (NLTK) [4] est utilisée nativement par *Chatterbot*. Ce préprocessing est réalisé afin d'éviter d'avoir des phrases qui d'un point de vue syntaxique se ressemblent, mais qui en termes sémantiques n'ont rien en commun.

Avec cet algorithme, une liste de questions similaires va ressortir. En effet, il s'agira de toutes les questions encodées et similaires par rapport à la question d'un étudiant. Chaque question aura un pourcentage de similarité avec la

4. https://www.nltk.org/

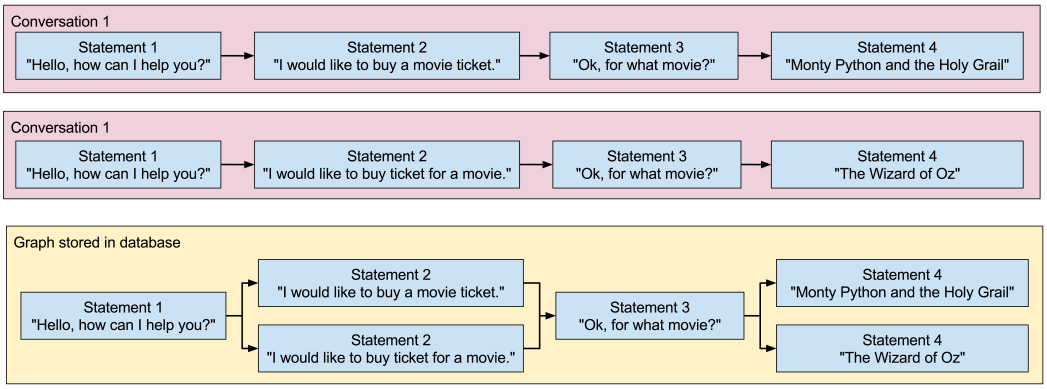


FIGURE 12 – Exemple de stockage des questions/réponses via un entraînement. En rose les questions/réponses entraînées et en jaune le stockage/matching en base de données [Cox, 2019].

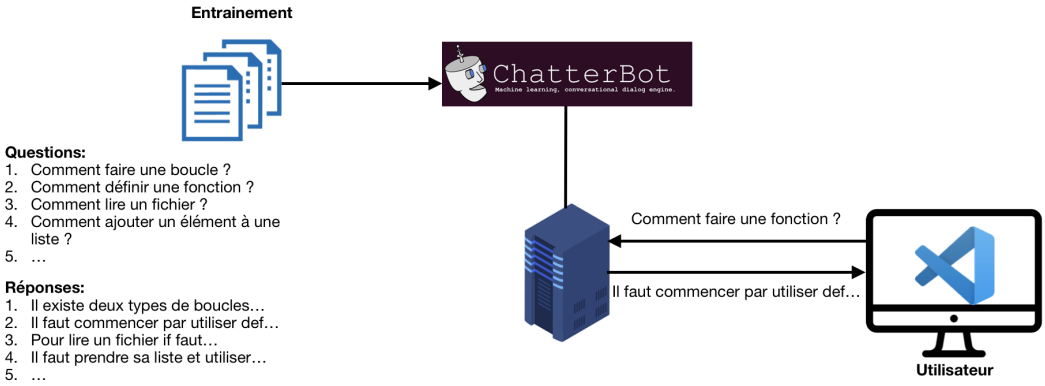


FIGURE 13 – Exemple d'un cas d'utilisation du chatbot sur base d'un entraînement de questions/réponses.

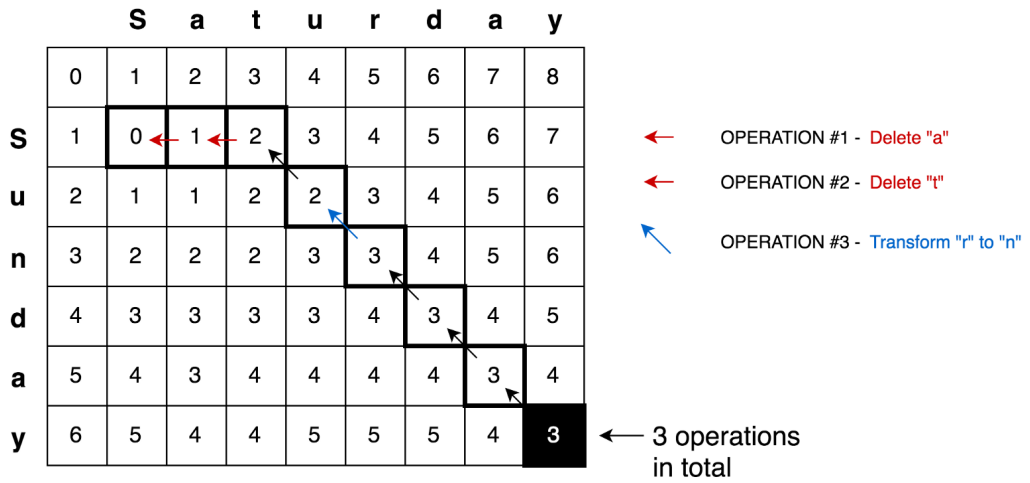


FIGURE 14 – Exemple d'une substitution de la chaîne de caractères "Saturday" vers "Sunday" en appliquant l'algorithme de Levenshtein [Trekhleb, 2018].

question de l'étudiant. Par la suite, en définissant un seuil (threshold) minimal au préalable, l'algorithme regarde si dans cette liste il existe une ou plusieurs questions similaires supérieures à ce seuil.

- Dans le cas où il y a plusieurs questions similaires, cela veut dire qu'il existe des questions encodées assez proches de la question posée par l'étudiant. La question ayant obtenu le score le plus haut sera dès lors choisie et donc, sa réponse associée.

- Dans le cas où il n'y en a qu'une seule supérieure au seuil, ce sera celle-ci qui sera retenue avec sa réponse associée.

- Dans le cas où, dans la liste, il n'y a aucune question supérieure au seuil, un message automatique qui stipule que la question n'a pas été comprise sera donné. La question laissée sans réponse sera sauvegardée dans la base de données afin de garder une trace de celle-ci. Grâce à cette sauvegarde, les professeurs et assistants seront amenés à découvrir quelles sont les questions posées par les étudiants qui n'ont pas encore été anticipées. En plus de cela, il sera possible rapidement d'enregistrer une réponse à la question pour réaliser un entraînement sur le chatbot et l'enrichir avec des questions d'étudiants.

Le chatbot est l'élément visible principal de la solution. Il est le point de contact avec les étudiants, c'est lui qui communique les réponses aux questions ainsi que le feedback sur le code. En effet, en plus de répondre aux questions (figure 15), celui-ci contient un interpréteur et permet de donner des informations sur la qualité du code.

Prenons l'exemple d'un étudiant qui, pendant la réalisation d'un exercice, est amené à écrire un code contenant une boucle. Celui-ci va écrire le code suivant :

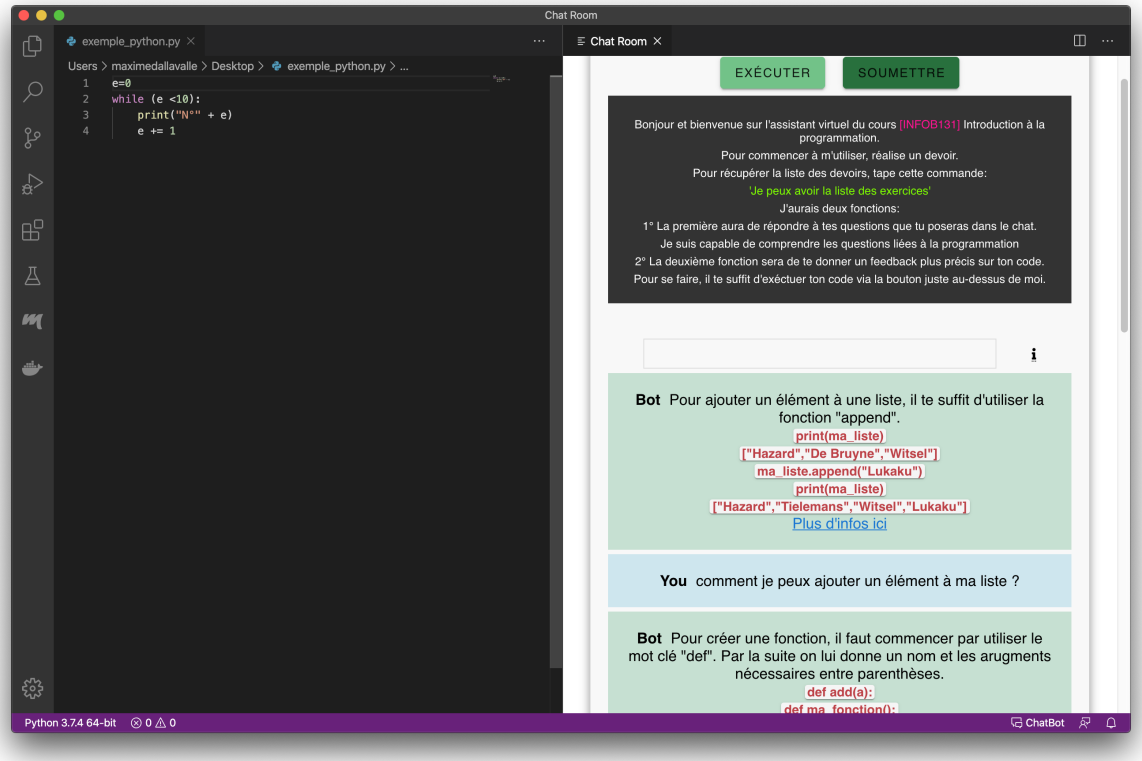


FIGURE 15 – Exemple de questions posées au chatbot par un étudiant dans le cadre d'une réalisation d'un exercice.

```
my_tab = []
for i in range(0,5):
print(i)
```

Durant toute la réalisation de l'exercice, il pourra poser des questions au bot, mais il pourra également tester son code. S'il venait à le faire tester, il en ressortirait les informations suivantes :

```
>NomDuFichier:3:1: IndentationError: expected an indented block
```

Le feedback reste similaire à un interpréteur sur le fond, mais sur la forme celui-ci est présenté de manière différente (figure 16). Ce feedback pourra également être traduit au préalable par un administrateur de la plateforme pour éviter les confusions linguistiques et ainsi modifier le sens du feedback. Au sujet de la forme, le feedback est mis en forme de manière très simple (en forme de liste). Il a été abordé auparavant que la manière dont est perçu un feedback est très importante sur la motivation de l'étudiant. L'idée du bot est

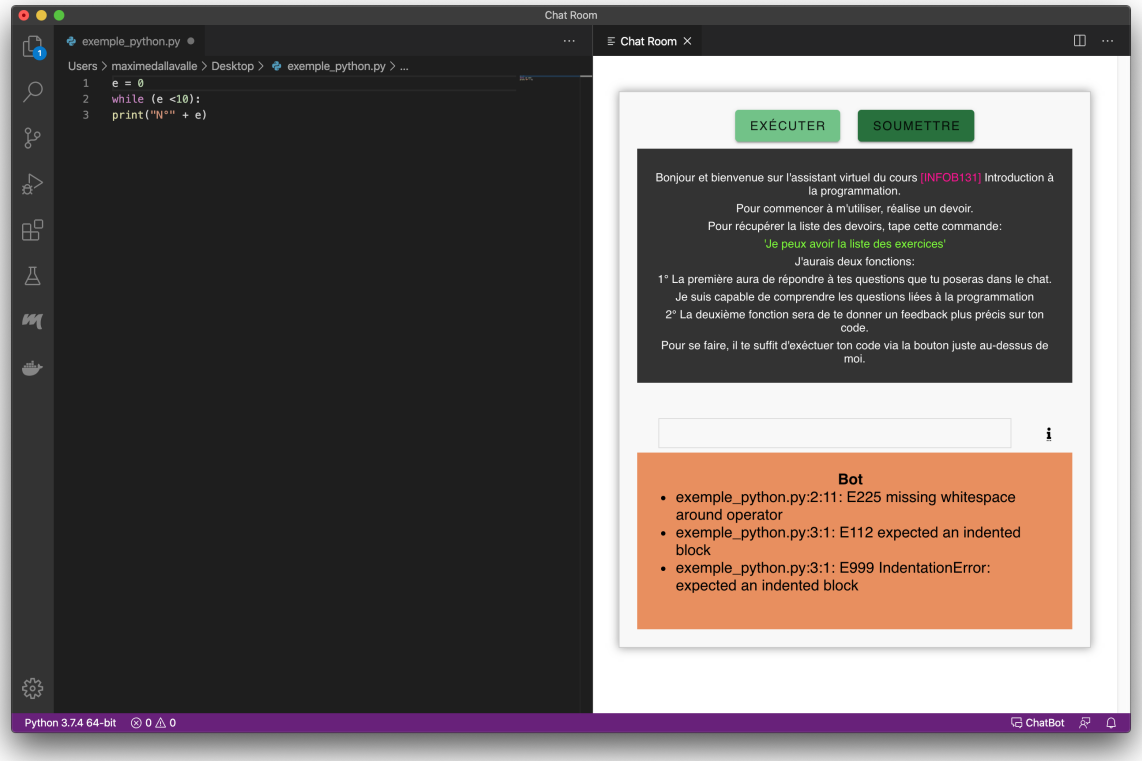


FIGURE 16 – Exemple d'un feedback par rapport à l'exécution du code d'un étudiant.

d'encourager l'étudiant à résoudre ses erreurs et à continuer sur sa lancée. En interprétant le code de l'étudiant directement dans le bot, cela permet d'avoir une mainmise sur le « comment » l'information sera représentée vis-à-vis de l'étudiant.

La conception du chatbot a donc été pensée pour répondre aux problèmes que les étudiants rencontrent dans les différentes phases d'un exercice (section 2.1.3). Précédemment, on a remarqué que le manque d'assistance peut affecter l'étudiant dans son apprentissage, et ce, de manière négative. Dès lors, le bot a été élaboré pour répondre aux questions qu'un étudiant se pose sans devoir chercher indéfiniment où poser la question. Plus encore, si un étudiant rencontre des difficultés, le fait qu'il puisse exécuter son code pour obtenir un feedback renforce le but d'assistance du chatbot. En effet, en procédant de cette manière, l'étudiant n'aura plus qu'un seul endroit à regarder pour obtenir l'ensemble des informations nécessaires à la réalisation d'un exercice. La charge cognitive est donc diminuée en procédant de cette manière. Le but est d'avoir un moyen d'accompagnement optimal de l'étudiant durant toutes les étapes de l'exercice.

Une dernière chose concernant la pédagogie du chatbot. Les feedbacks du bot ainsi que l'accompagnement que celui-ci procure durant toute la réalisation d'un exercice ont été abordés. Tout ceci peut permettre un changement de mentalité dans l'apprentissage de la programmation pour les étudiants. La section 2.2.2, qui introduit les termes "Fixed Mindset" et "Growth Mindset",

parle d'un changement d'une mentalité fixe vers une mentalité dite de croissance. Un accompagnement continu est fondamental pour la progression de l'étudiant et le chatbot va lui permettre de surmonter ses obstacles pour un meilleur apprentissage et une réduction des abandons. Les feedbacks proposés offrent une réponse correcte formulée de façon positive pour l'étudiant. La complémentarité de la plateforme avec les professeurs et assistants peut permettre un meilleur soutien ce qui évite au maximum le décrochage des étudiants. Cette complémentarité sera abordée dans la section suivante.

Concernant les détails techniques de l'interface du chatbot, celle-ci a été reprise d'un code JavaScript [5] pour un chatbot. Ce code a été amélioré afin d'être capable d'interagir avec des questions parlant de langage de programmation, comme pour l'affichage du code. Ce code a été choisi, car il disposait d'une interface très claire et il était relativement aisé de modifier le contenu. En fin de compte, cette partie n'était que visuelle, puisque tout le processus de recherche de questions similaires, explicité précédemment se fait avec *Chatterbot* au niveau du back-end.

L'intégration du chatbot a été réalisée sur VSCode sous forme d'une extension. À chaque fois que l'étudiant est amené à réaliser un exercice, il suffit de lancer le chatbot via un bouton sur VSCode. Il a déjà été expliqué dans les sections précédentes que l'interaction du bot avec l'étudiant se veut claire pour éviter de créer une trop grosse charge cognitive. L'extension vient charger une IFrame [6] où se trouve le chatbot qui est hébergé sur une plateforme Web. Cette technique permet de découper complètement l'extension et concentrer le code sur une plateforme Web afin de pouvoir facilement l'éditer dans le futur.

Il est intéressant de souligner que l'interface de VSCode peut être réduite au plus simple : une page pour coder d'un côté et le chatbot de l'autre. Cela permet à l'étudiant de rester pleinement concentré sur son exercice. Concernant l'installation de ce plugin par les étudiants, il leur suffit de se rendre sur la page Web [7] de l'extension et de cliquer sur *installer*.

Pour terminer, il est intéressant de montrer que la force du chatbot est sa continuité dans le temps. Lorsqu'un étudiant posera une question à laquelle le bot ne peut répondre, celle-ci sera sauvegardée pour qu'un professeur ou un assistant puisse y apporter une réponse. De ce fait, les prochains étudiants qui poseront la même question (ou une question y ressemblant) obtiendront une réponse. L'idée est donc de construire un chatbot évolutif dans le temps avec une intégration continue du feedback des étudiants sur l'outil.

En Bref ! : Chatbot

Le chatbot est la pièce centrale de la solution. Celui-ci a pour objectif d'accompagner l'étudiant pendant toute la réalisation d'un exercice. Le chatbot est capable d'assister l'étudiant grâce à sa base de connais-

5. https://github.com/ddsky/chatbot
6. https://developer.mozilla.org/fr/docs/Web/HTML/Element/iframe
7. https://marketplace.visualstudio.com/items?itemName=maxdal.chat-room

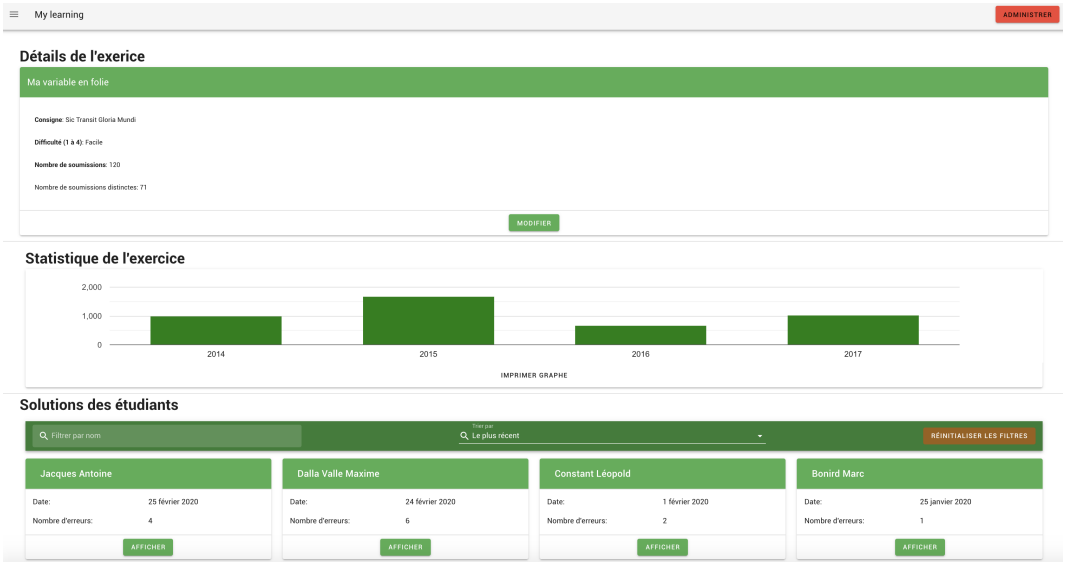


FIGURE 17 – Interface d'administration pour un exercice.

sances sur le langage de programmation Python. Lorsqu'un étudiant vient lui poser une question, celui-ci utilise l'algorithme de *Levenshtein* pour lui donner une réponse pertinente. Le chatbot s'assure que l'étudiant obtient une réponse, dans le cas contraire, la question de l'étudiant est sauvegardée afin de permettre d'y apporter une réponse ultérieurement.

3.3.2 Interface pédagogique

En parallèle avec le chatbot, une plateforme Web a été créée afin de synthétiser les informations des exercices réalisés par les étudiants. Cette plateforme est, aujourd'hui, exclusivement adressée aux professeurs et assistants. Le but de celle-ci est d'offrir aux professeurs et assistants des moyens visuels pour savoir où se situent les étudiants dans leur apprentissage. Cela peut passer par des statistiques ou simplement par la visualisation d'un code d'un étudiant pour un exercice (figure 17). L'intérêt est de pouvoir identifier en un coup d'oeil les problèmes des étudiants. À l'heure actuelle, il est uniquement possible de savoir quelles sont les questions et les erreurs les plus fréquemment posées et rencontrées par les étudiants. Il est possible de visualiser un code en détail pour obtenir un exemple d'erreurs faites par les étudiants.

Pour connaître le nombre de fois qu'une réponse a été posée, l'*algorithme de Levenshtein* est utilisé (celui déjà employé pour le bot). Lorsqu'un étudiant est amené à poser une question au bot, s'il y a une correspondance avec une question encodée, une information sera ajoutée à cette question stockée dans la base de données afin de savoir si cette question a déjà été demandée, et surtout à quelle fréquence. Au final il est possible d'obtenir pour chaque question entraînée le nombre de fois que celle-ci a été posée par les étudiants à travers le chatbot sur VSCode. Le principe est exactement le même pour une question qui

2 QUESTION TYPE (PAS DE PONCTUATION,MAJUSCULE)	POSSÈDE UNE RÉPONSE ASSOCIÉE	1 NOMBRE DE FOIS QUE LA QUESTION A ÉTÉ POSÉE
comment lire un fichier	✓	46
c'est quoi un chemin	✓	23
comment ecrire dans un fichier	✓	15
acceder a un fichier	✓	9
comment recuperer les informations d'un fichier	✓	2

FIGURE 18 – Questions posées par les étudiants ayant eu un matching.

n'a pas de correspondance (plus de détails dans la section 3.3.1). Ceci constitue la base des statistiques fournie par le système.

L'autre partie concerne les erreurs faites par les étudiants. Le système proposé est relativement simple et permet d'avoir un aperçu des erreurs que les étudiants peuvent être amenés à faire durant la réalisation d'un exercice. Comme vu auparavant, à chaque fois qu'un étudiant exécute son code celui-ci est analysé avant qu'un retour ne lui soit donné par le chatbot. Par ailleurs, le code est enregistré et est mis à disposition des professeurs et assistants pour consulter les réalisations des étudiants. Vu la probabilité que les responsables de la plateforme se retrouvent vite submergés par les nombreux codes d'étudiants, deux solutions ont été proposées pour pallier cette problématique :

1. Définir une soumission d'un exercice comme « finale ». Lorsque l'on regarde la figure 15, on constate la présence de deux boutons. L'un proposant à l'étudiant de simplement exécuter son code afin d'obtenir un feedback sur celui-ci et l'autre lui donnant la possibilité de rendre définitive la solution qu'il propose à l'exercice associé. De cette manière, les codes des étudiants seront filtrés par les professeurs et assistants entre les simples exécutions (à savoir les tentatives multiples de résolution de l'étudiant) et les soumissions finales (à savoir la solution définitive de l'étudiant). Toutefois, cette méthode ne règle pas entièrement la question de la surcharge des codes des étudiants puisqu'environ 200 étudiants sont inscrits au cours *[INFOB131] Introduction à la programmation.*

2. Cela conduit à la deuxième solution en lien avec la partie statistique. À chaque exécution et soumission, les erreurs commises par les étudiants sont collectées afin de les représenter sous forme d'un graphique. Plus précisément, il est question d'un graphique en bâtons afin de distinguer rapidement le nombre d'erreurs commises en fonction de son type (figure 19). D'autres graphiques sont également proposés pour rapidement identifier où se trouvent les étudiants dans la réalisation des exercices. Les améliorations possibles du système sont détaillées dans la partie 5.3.

Dans la section précédente, la notion de complémentarité a été abordée. L'idée derrière cette plateforme Web est de fournir un maximum d'informations concernant le niveau des étudiants à un instant « t » de manière succincte et rapide. Le fait de posséder des statistiques sur le nombre d'erreurs, de réalisations, etc., permet aux professeurs ou assistants de savoir en un clin d'oeil où en sont les étudiants dans la réalisation d'un exercice, mais également de savoir quelles sont les erreurs les plus fréquentes. Grâce à ces informations, il semble désormais plus aisé de cibler les problèmes des étudiants et donc, d'arriver à les résoudre plus rapidement en séance de TP. Identifier les étudiants

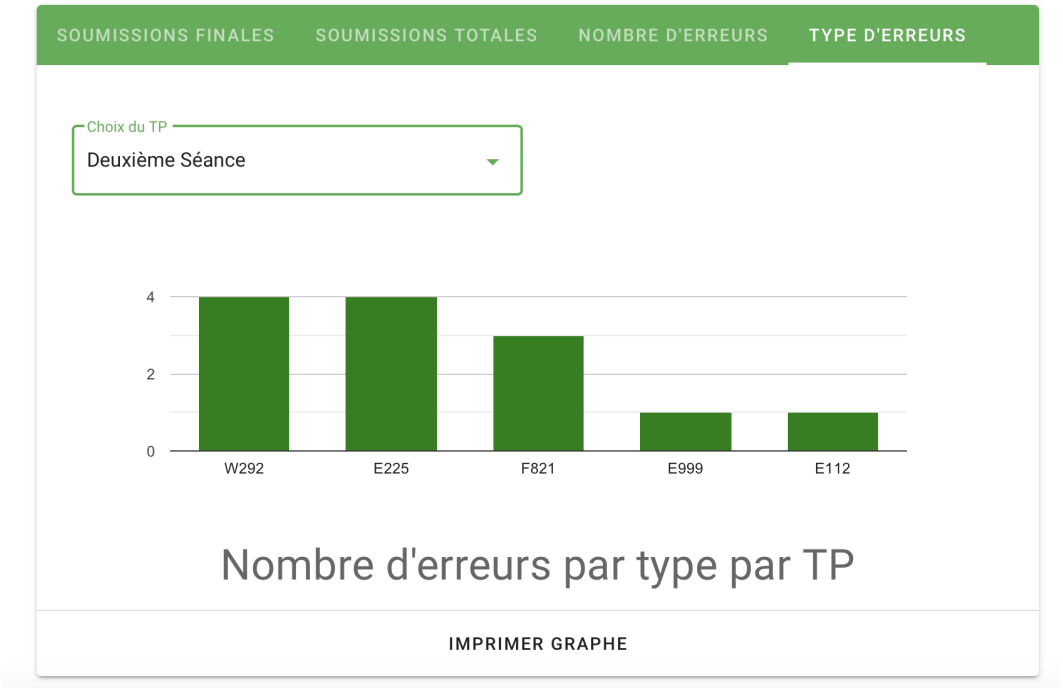


FIGURE 19 – Nombre d'erreurs par code d'erreur.

qui éprouvent des difficultés ou qui commettent des erreurs classiques, permet d'éviter un décrochage de ces étudiants. D'un point de vue pédagogique, la solution propose un outil qui permet de cibler les problèmes des étudiants et ainsi elle permet de fournir un support plus rapide et plus ciblé de la part du corps enseignant.

Au niveau technique, le "front-end" de cette interface pédagogique a été conçu grâce au framework *VueJS*[8] avec Vuetify[9] comme composant. Tout le site Web se base sur le back-end réalisé en *Django*[10]. Le choix s'est porté sur ce dernier, car il était en accord avec nos contraintes :

- développement en Python qui est un langage enseigné à la faculté d'informatique de Namur
- il met l'accent sur un développement rapide (correspondance avec le temps accordé par le stage)

En Bref ! : Interface pédagogique

La plateforme Web est l'endroit où le corps enseignant peut se rendre pour observer rapidement les difficultés des étudiants, mais également pour paramétrer le chatbot. Ce site a pour objectif d'être complémentaire avec les outils déjà existants et donc de travailler en

8. https://vuejs.org/
9. https://vuetifyjs.com/fr-FR/
10. https://www.djangoproject.com/

collaboration avec les professeurs et assistants dans leurs missions pédagogiques.

3.3.3 Serveur

Le serveur est l'élément central du projet. C'est lui qui permet l'exécution des codes des étudiants.

Afin de proposer une solution performante qui permet de satisfaire aux différentes contraintes possibles pour ce genre de plateforme, il était nécessaire que l'architecture soit bien segmentée et sécurisée. La figure 20 reprend tous les éléments qui composent la solution. Cette vision de l'architecture peut paraître une vue de haut niveau (haut niveau d'abstraction). Chacun des modules sera d'abord présenté de manière individuelle, afin que l'aspect technique de la solution puisse être pleinement exprimé et compris. Ensuite, le coeur du fonctionnement de chacun de ces sous-modules sera abordé au travers des diagrammes de séquence démontrant l'enchaînement et les relations qui existent entre nos modules, mais surtout le caractère performant de la solution. Ainsi, la complexité intrinsèque de la solution pourra être décomposée et le lecteur pourra mieux se concentrer sur ces relations. En procédant de cette façon, le lecteur pourra dès lors s'imprégner de l'architecture et imaginer de possibles nouvelles fonctionnalités à la solution.

La figure 20 peut être divisée en cinq parties. Ces parties sont numérotées dans le cadre les délimitant. Vous pouvez remarquer que l'ensemble de ces parties est dockerisé (empaqueté une application et ses dépendances dans un conteneur isolé). Docker a été choisi pour différentes raisons. Parmi celles-ci :

- Simplification de l'installation de l'architecture que ce soit pour un administrateur système ou pour un nouveau développeur
- Meilleure séparation de l'architecture en service ayant une fonctionnalité unique. Ceci a comme avantage une meilleure évolutivité de la solution. La mise à jour d'une librairie dans un conteneur n'a pas d'impact sur le reste du système.
- Cela rend le déploiement plus facile, car le système en développement ou en déploiement est sensiblement le même.
- Contrôle facile des communications inter conteneur et limite des accès externes à ceux-ci.

(1) Serveur Web applicatif : *Django*

Django[11] est un framework Python de développement Web. Par le passé, une certaine maîtrise et des connaissances avaient été acquises à travers la réalisation de projets avec Django. Puisque le langage Python est enseigné à la

11. https://www.djangoproject.com/

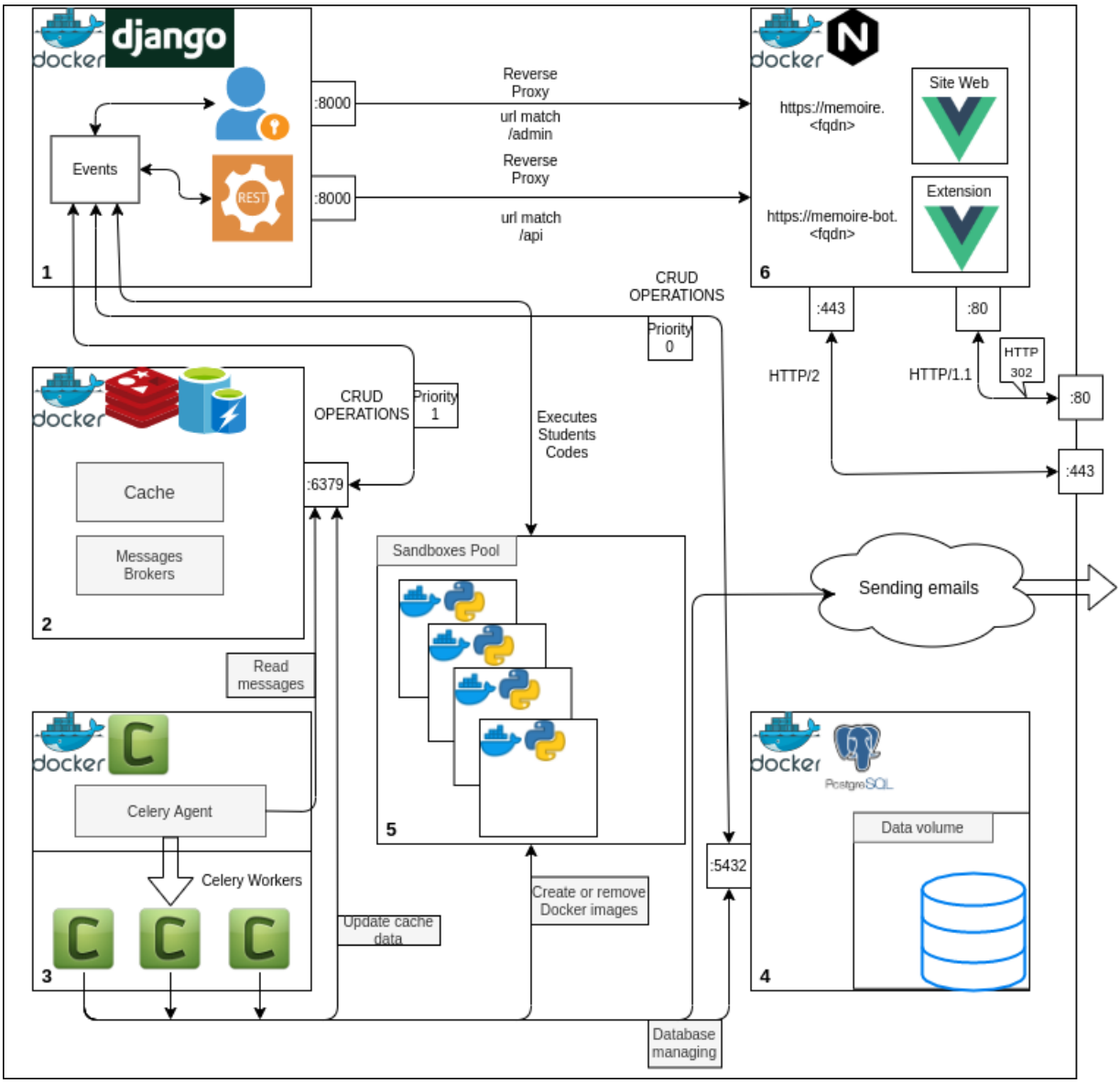


FIGURE 20 – Architecture

faculté, ce choix semble être judicieux. Ce framework permet de répondre à tous les besoins d'un serveur back-end. Tout d'abord, la mise en place de la gestion des utilisateurs s'est établie grâce à une double authentification :

1. Il y a tout d'abord l'authentification propre à *Django* (le moteur *back-end* et par défaut). Cette authentification est uniquement utilisée pour accéder au panel administrateur généré par *Django*. Le choix a été fait de garder le panel d'administration, car il permettait nativement d'effectuer toutes les opérations de création, modification, mise à jour et suppression (Create Read Update Delete (CRUD)) sur les instances définies et utilisées par le système.

2. Ensuite, la seconde authentification est de type *JSON Web Token (JWT)*. Ce standard ouvert défini dans le RFC 7519 [12] permet de créer un jeton qui identifie les utilisateurs et donc leurs permissions, mais également sa propre durée de validité. Ce token est signé par un algorithme. Dans le cas présent, c'est un *Hash-based Message Authentication Code (HMAC)* [13] basé sur SHA-256 [14] qui est utilisé. Cette combinaison d'algorithmes de hash est définie dans le RFC 4868 [15]. Ce sont des solutions éprouvées. Il est également à noter que le système pourrait très bien fonctionner avec d'autres algorithmes de hashage (SHA-384, SHA-512) ou même RSA, mais des bénéfices (au niveau de la bande passante et de la sécurité) qui seraient apportés par ces changements ne sont pas évidents.

Toujours sur l'aspect sécuritaire, le système permet de vérifier l'identité des utilisateurs de manière double. Cela signifie qu'il est possible de s'authentifier via deux systèmes différents et recevoir au final le token JWT. Le premier est une authentification dite classique se basant une comparaison de hash. L'algorithme de hash choisi est considéré comme solide : Argon2 [16] ; il a été récompensé lors du *Password Hashing Competition* [17] de 2015.

Django fournit un Object Relational Mapping (ORM). Ce système joue le rôle d'une interface entre nos entités, implémentées de manière orientée objet et le Système de Gestion de Bases de Données Relationnelles. Cela fournit donc une couche d'abstraction qui permet de se passer de requêtes SQL. Dans l'implémentation, les objets qui héritent de la classe "Model" de *Django* permettent de réaliser toutes les opérations possibles sur les données.

Django possède également le concept de "view", permettant d'encapsuler la logique nécessaire afin de renvoyer à un utilisateur une réponse à sa requête. L'utilisation de ce concept a été spécialisée en utilisant l'architecture REST [18]. Cette architecture est très populaire et répond parfaitement aux contraintes de la solution. Concrètement, c'est sur base d'une librairie communautaire

12. https://tools.ietf.org/html/rfc7519
13. https://tools.ietf.org/html/rfc2104
14. https://csrc.nist.gov/publications/detail/fips/180/4/final
15. https://tools.ietf.org/html/rfc4868
16. https://en.wikipedia.org/wiki/Argon2
17. https://password-hashing.net/
18. https://en.wikipedia.org/wiki/Representational_state_transfer

reconnue (*Django REST framework*[19]) que l'API REST a été construite. Cette boîte à outils a facilement permis de définir des classes reliées à chacun de nos objets ; certaines méthodes de celles-ci implémentent les opérations liées aux requêtes HTTP. Par exemple, la méthode "*retrieve*" sera automatiquement associée à une requête GET[20].

La mise en place d'une API REST nécessite pour le système une grande manipulation de structures de données JSON. L'implémentation proposée par Python pour manipuler ce type de structure n'est pas du tout optimisée, que cela soit au niveau du temps ou de la charge en mémoire. Une librairie externe bien plus performante a été utilisée : *orjson*[21]. Ce choix n'implique aucun changement dans l'écriture de l'API REST. Il suffit de configurer *Django REST framework* afin qu'il utilise les classes appropriées.

L'architecture REST implique également une gestion du cache. Celle-ci a été mise en place et sera précisée dans les parties détaillant la base de données à accès rapide et la file de tâches distribuées. En effet, le choix a été fait de remplacer ou de compléter certaines fonctionnalités de base de *Django* afin de proposer des améliorations.

Ce serveur Web est propulsé grâce à un serveur *Daphne*[22] qui est l'implémentation recommandée d'un Asynchronous Server Gateway Interface (ASGI)[23]. Cette solution est suffisamment performante et prend en compte le fait que le serveur *Django* sera rendu accessible aux utilisateurs à travers un *reverse proxy*.

Tous les éléments cités précédemment se retrouvent au sein d'un conteneur *Docker* basé sur l'image *Python* officielle dans sa version slim[24].

(2) Base de données à accès rapide : *Redis*

Redis est une base de données de type clé-valeur. Ce système de gestion de base de données NOSQL fournit de hautes performances grâce à son stockage intégralement situé dans la mémoire RAM et permet donc d'éviter les accès disques. Elle est utilisée pour deux raisons :

1. Tout d'abord en tant que cache du serveur *Django*. En effet, un bon nombre de données sont utilisées de manières communes par tous les utilisateurs d'une plateforme. Certaines de ces données n'ont pas tendance à changer de manière fréquente. En les gardant dans le cache de l'application, on évite ainsi des manipulations plus lentes de la part de la base de données SQL et permet ainsi d'avoir une API plus réactive pour ces données en cache. Celles-ci ont une certaine durée de validité, configurée à une heure, mais cette valeur pourrait très bien être adaptée tout en prenant

19. https://www.django-rest-framework.org/
20. https://developer.mozilla.org/fr/docs/web/HTTP/M%C3%A9thode/GET
21. https://github.com/ijl/orjson
22. https://github.com/django/daphne
23. https://github.com/django/asgiref/blob/master/specs/asgi.rst
24. https://hub.docker.com/_/python

en compte la présence d'un système asynchrone de mise à jour du cache en cas de modifications des données s'y rapportant.

2. Certaines tâches du système pourraient prendre du temps et avoir un impact sur les performances du serveur applicatif. Ces tâches coûteuses sont rédigées sous la forme de messages stockés dans la BD *Redis*. Ces messages sont ainsi placés dans une file où ils attendent d'être traités par un travailleur (worker), en l'occurrence *Celery*. Ainsi *Redis* se place uniquement en tant qu'intermédiaire pour ces messages et permet donc le passage d'informations entre le serveur *Django* et *Redis* explicités plus bas.

La mise en place de cette Base de Données se fait facilement en travaillant avec l'image officielle *Docker* de *Redis* dans sa version *Alpine*[25] et en exposant son port (*6379*) au réseau interne de l'application.

(3) File de tâches distribuées : *Celery*

Celery[26] est une file de tâches asynchrones. En faisant passer des messages à travers le cache *Redis*, le serveur applicatif *Django* se débarrasse de tâches chronophages pour lui et se concentre ainsi sur la réactivité de son API. Parmi ces tâches, on peut citer l'envoi de mails, la construction des images *Docker* utilisées dans le système, certaines écritures dans la base de données SQL ou encore la mise à jour du cache. Toutes ces tâches ont en commun, pour le système, de ne pas avoir une utilisation instantanée ou de prendre un temps relativement conséquent.

Concrètement, *Celery* est un agent de messages (message broker) qui va aller lire les tâches écrites par le serveur *Django* au sein du *Redis* et mettre ses tâches dans une file d'attente où elles seront traitées par des workers (processus). Ce type de système permet d'éviter de créer des sous-processus au sein du serveur *Django*.

(4) Base de données SQL : PostgreSQL

PostgreSQL a été choisi comme Système de Gestion de Bases de Données.

Ce système est de type relationnel mais également de type objet. Grâce à ce dernier aspect, *PostgreSQL* est également appelé Système de Gestion de Bases de Données Relationnelles-Objets (SGBDRO). Ce système fournit une fonctionnalité particulièrement intéressante : le stockage d'objets JSON. Ce format d'objets est particulièrement utilisé dans le monde des services Web. En effet, cela place *PostgreSQL* à la frontière du monde NOSQL et la flexibilité donnée par celui-ci est élevée, possédant ainsi le « meilleur des deux mondes ».

25. https://hub.docker.com/_/redis/
26. http://www.celeryproject.org/

PostgreSQL est entièrement supporté par *Django* et sa configuration s'est faite sur base de son image *Docker* dans sa version *Alpine*[27]. Afin de garantir la persistance de nos données, un volume *Docker* externe[28] a été utilisé.

(5) Serveur Web : Nginx

Afin que la plateforme puisse échanger des mots de passe et d'autres informations confidentielles, il était important de mettre en place un serveur Web performant et chiffré. En tant que serveur Web le plus populaire[29], *Nginx* a été sélectionné. Ce serveur Web propulse donc deux versions statiques (construites à partir du code *VueJs*) de la plateforme Web : une première classique et une seconde adaptée à l'affichage dans une IFrame (utilisée par l'extension VSCode).

À titre indicatif et afin d'étayer le fait que ce serveur Web est correctement chiffré, le score de 100/100 sur le site de référence SSLLabs[30] a été obtenu. Notons toutefois que l'entête HTTP *X-Frame-Options*[31] n'a pas pu être activée pour le site utilisé par l'extension.

Il convient de souligner que le serveur *Nginx* n'est pas utilisé lors du développement, mais bien le serveur de développement par défaut de *VueJs*[32].

Enfin, l'image *Docker* utilisée est l'officielle dans sa version *Alpine*[33].

(6) Sandboxes Pool

Pour la réalisation du logiciel, les codes des étudiants sont exécutés sur un serveur distant. Cela a pour avantage de garder un historique des exécutions des étudiants et par conséquent également les étapes que ces étudiants ont réalisées afin de réussir un exercice. En outre, cela leur évite d'installer et de configurer un environnement de développement en accord avec le projet (version de Python, librairies externes, fichiers utilisés spécifiquement pour l'exercice).

Cependant, exécuter des codes à partir de sources inconnues peut s'avérer dangereux. Il faut s'assurer qu'aucun impact sur le serveur n'est possible et que l'exécution d'un exercice ne puisse altérer son état. Cette isolation pourrait être réalisée grâce à la virtualisation. Néanmoins, elle empêcherait de fournir un logiciel « clé en main » et c'est la raison pour laquelle ce type d'isolation a été écarté.

27. https://hub.docker.com/_/postgres
28. https://docs.docker.com/storage/volumes/
29. https://news.netcraft.com/archives/2019/04/22/april-2019-web-server-survey.html
30. https://www.ssllabs.com/ssltest/analyze.html?d=memoire.jacquant.be&hideResults=on&latest
31. https://developer.mozilla.org/fr/docs/web/HTTP/Headers/X-Frame-Options
32. https://cli.vuejs.org/guide/cli-service.html#vue-cli-service-serve
33. https://hub.docker.com/_/nginx

Tout d'abord, des logiciels comme *AppArmor*[34] et *SELinux*[35] avaient été envisagés. Ces solutions permettent d'associer à chaque programme un profil de sécurité avec des règles prédéfinies. Ainsi, après avoir défini les règles associées à la version *Python* d'un exercice, il aurait été possible d'exécuter les codes des étudiants dans des sous-processus de manière sécurisée. Pourtant cette solution a été abandonnée, malgré son fonctionnement, au vu des nombreuses contraintes. Parmi celles-ci, il est possible de mentionner la configuration qui aurait nécessité une personne suffisamment expérimentée avec les systèmes *Linux* ou encore que cette solution oblige à tourner uniquement sous *Linux*. Par ailleurs, ces logiciels nécessitent de charger des modules dans le noyau *Linux* et puisque *Docker* hérite des modules du noyau *Linux* hôte, il est obligatoire que ces applications soient installées en « dur » sur le serveur hôte pour fonctionner au sein du conteneur *Django*. Cette solution d'isolation a été jugée trop compliquée à implémenter ou à maintenir dans le futur.

L'isolation a donc été réalisée sur base de conteneurs *Docker*. Cette solution est plus facile à mettre en place et permet un contrôle plus simple des ressources utilisées lors de l'exécution d'un code en provenance d'un étudiant. En effet, le temps d'exécution, la quantité de mémoire et d'autres contraintes peuvent être fixés afin d'éviter qu'un étudiant ayant par exemple implémenté une boucle infinie, bloque le système ou utilise plus de ressources que nécessaire. Malgré le fait que *Docker* fournisse nativement une certaine isolation, il a fallu être particulièrement vigilant à ce qu'au travers de cette exécution de code l'utilisateur malicieux ne puisse pas avoir un accès aux fichiers du serveur hôte ou même pouvoir ouvrir des ports. Face à ces risques, *Docker* seul n'apporte pas de réponse, mais permet néanmoins d'avoir une solution utilisable par les développeurs. Cette solution est cependant moins performante que la première, car il est nécessaire d'initialiser et de lancer un conteneur *Docker* avant de pouvoir réellement exécuter le code de l'étudiant.

Comme bien souvent en informatique, la solution optimale se retrouve dans la composition de solutions afin de trouver le meilleur compromis : configurer correctement une application comme *AppArmor* ou *SELinux*, mais uniquement pour *Docker*, pas nécessairement spécifique pour *Python*. La documentation de *Docker* est complète[36] concernant les moyens d'augmenter la sécurité du système. Ces recommandations doivent être mises en application lors du déploiement.

En Bref ! : Serveur

L'architecture est une composition de conteneurs *Docker*, chacun ayant sa propre fonction. Ceux-ci communiquent entre eux afin de récupérer des données des bases de données, de réaliser des tâches lourdes, de mettre à jour les bases de données, etc. Ensemble, ils

34. https://gitlab.com/apparmor/apparmor
35. https://github.com/SELinuxProject/selinux
36. https://docs.docker.com/engine/security/security/

contribuent au bon fonctionnement du système en étant axés sur les performances globales.

3.3.4 Interactions de nos composants

Dans la section précédente, l'architecture a été décomposée en différentes unités. Ces unités communiquent entre elles afin de fournir des interactions plus complexes. Seules certaines de ces interactions ont été développées. Ces dépendances entre les différentes composantes de notre architecture sont à chaque fois générées sur base d'input d'un utilisateur. Le système réagit afin de produire le résultat escompté et en impliquant les différents modules nécessaires. Seuls les inputs qui permettent d'illustrer la bonne compréhension de l'architecture et des relations existantes entre les composants sont détaillés.

Un utilisateur demande des données avec ou sans cache

Dans ce type de cas de figure, un utilisateur pose tout simplement une question spécifique au bot. Cette action de la part de l'utilisateur est en réalité plus spécifique qu'un simple appel API pour récupérer des données. En effet, le fonctionnement du cache et les étapes évitées avec celui-ci seront valables pour toutes les interactions impliquant le chargement des données de notre système.

Dans la figure 21, on peut comparer les deux situations. L'objectif du cache est de récupérer plus rapidement la réponse souhaitée. En effet, une requête réalisée dans une base de données clé-valeur (comme *Redis*) et utilisant de la mémoire RAM peut être jusqu'à dix fois plus rapide.

Ainsi à chaque fois qu'une requête est réalisée, il faut vérifier tout d'abord si ces données peuvent être accédées depuis le cache. Cela nécessite un test et donc une connexion au cache à chaque appel. Si le cache contient les données, il les retourne et il ne reste à l'API qu'à les formater pour les remettre à l'utilisateur. Ce scénario correspond à la figure 21b. Une illustration avec les secteurs actifs de l'architecture peut être retrouvée en annexe avec la figure 30.

Au contraire,lorsque les données ne sont pas stockées dans le cache, il est nécessaire de réaliser deux actions. La figure 29 reprend le schéma de l'architecture (figure 20) en mettant l'accent sur les composants utilisés.

1. Les récupérer dans la base de données *PostgreSQL* via l'ORM *Django* ;
2. Mettre à jour le cache afin de profiter de la requête précédente pour celles futures sur la base de données SQL.

Ainsi, les utilisateurs profitent des requêtes des uns et des autres afin de fluidifier leur propre navigation dans le système. Cependant, les données mises dans le cache ont une durée de vie limitée. Cette dernière est d'une heure. Par conséquent, les données peu sollicitées ne prennent pas de la place dans la mémoire RAM inutilement, mais profitent également moins de ce système de cache. C'est un compromis à trouver. Notons également que malgré le fait que

certaines données soient fréquemment utilisées, il sera nécessaire de réaliser une requête sur la base de données SQL toutes les heures afin de remettre à jour les données dans le cache.

Requête sur le bot par un étudiant

Une des fonctionnalités principales de la solution est de fournir un bot pour répondre aux questions des étudiants. Dans cette situation, volontairement, aucun système de cache n'est mis en place. En effet, il est impossible d'anticiper à quel point les questions des utilisateurs auront des similitudes les unes des autres et donc, à quel point il est utile de mettre en place le système de cache. Ainsi selon la figure 22, on constate que la partie consacrée à *Chatterbot* va réaliser plusieurs actions. Tout d'abord, la question est ajoutée ou mise à jour au sein de la base de données (incrémentation du compteur associé au nombre de fois que chaque question est posée). Ensuite, grâce à la méthode détaillée dans la section 3.3.1, la meilleure réponse associée à la question est récupérée et va être renvoyée dans le format adéquat à l'utilisateur. Malgré le fait que dans la figure 22, la flèche allant du premier cadre lié à *Chatterbot* vers Database est unique, il s'agit en réalité de plusieurs requêtes sur la base de données (au minimum 2).

Sauvegarde d'un exercice par un utilisateur-administrateur

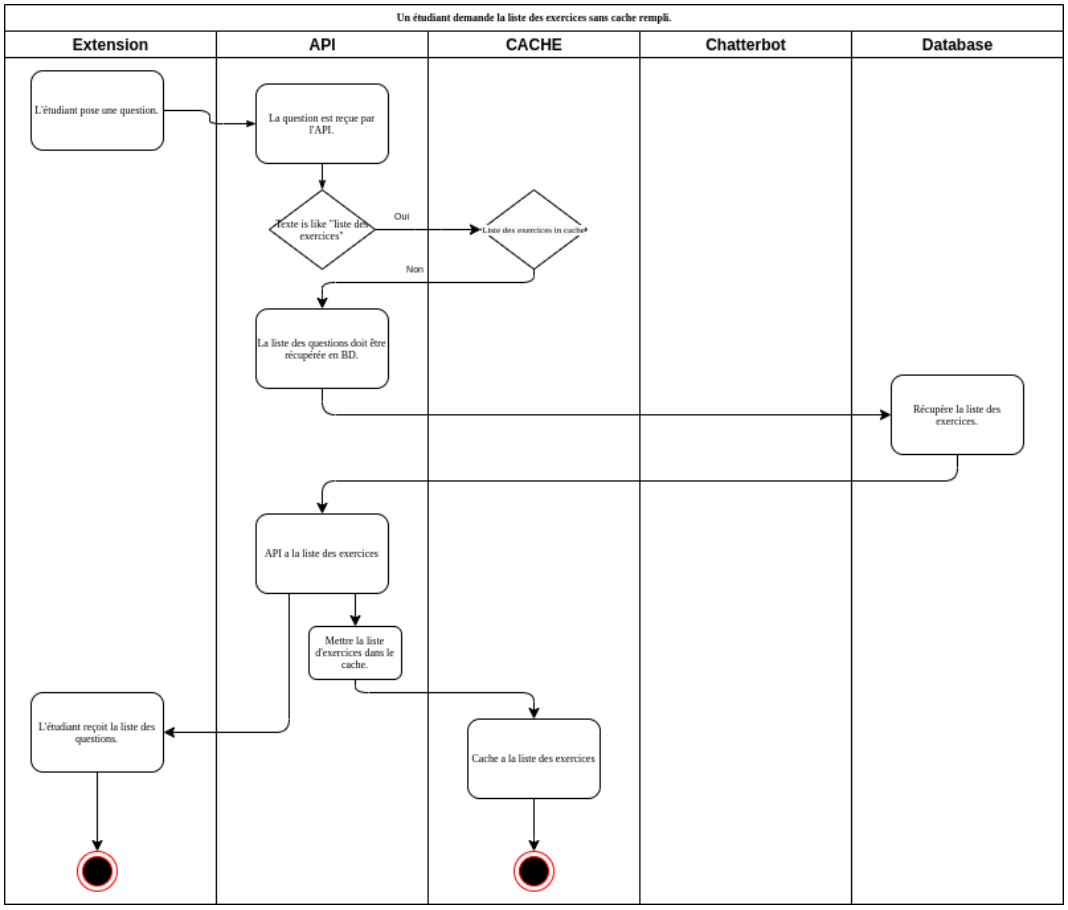
Cette situation est uniquement accessible par un utilisateur-administrateur. Lorsque les professeurs voudront réaliser les exercices des TP, ils utiliseront une interface bien spécifique et uniquement accessible par eux.

Comme représenté dans la figure 23, trois exécutions simultanées se produiront lors de la sauvegarde d'un exercice.

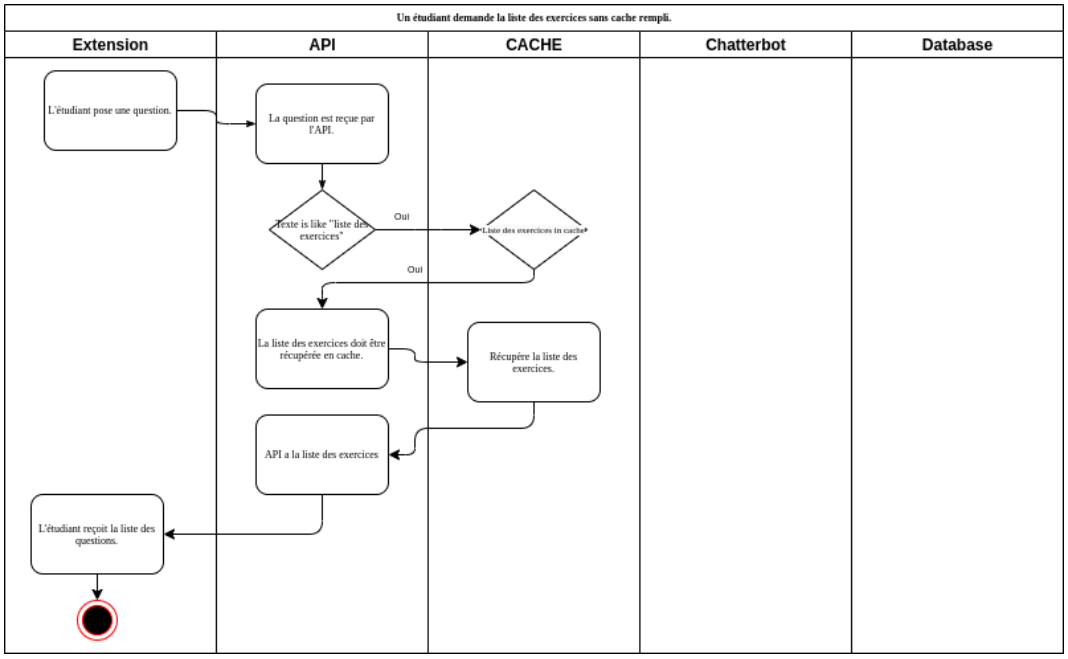
Tout d'abord, l'exercice créé sera sauvegardé dans la base de données *PostgreSQL*. Ensuite, une tâche sera créée afin que le *Celery* supprime les références liées aux exercices dans le cache afin de ne pas servir aux utilisateurs des données incomplètes ou erronées. Finalement, la tâche la plus longue en temps consiste en la création ou la mise à jour de l'image *Docker* associée à l'exercice. Cette tâche peut prendre un certain temps, car il pourrait être demandé de télécharger des ressources (comme des librairies) et de les fournir dans l'image. Il est toutefois à noter que *Docker* fournit un système de cache lors de la création d'images et par conséquent, toutes les étapes de la création d'une image *Docker* ne seront pas refaites à chaque fois[37].

Vu que la création de cette image prend du temps, qu'il est difficile à en estimer la durée exacte et qu'il n'est pas souhaitable de bloquer la navigation de l'utilisateur-administrateur, cette tâche sera confiée à un travailleur *Celery*. Une fois qu'il aura créé cette image *Docker*, l'objet exercice créé plus tôt dans les processus sera mis à jour afin d'avoir la référence à cette image.

37. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#leverage-build-cache



(a) Cache inactif ou non chargé lors d'une requête pour avoir la liste des exercices.



(b) Cache actif et chargé lors d'une requête pour avoir la liste des exercices.

FIGURE 21 – Comparaison entre une requête avec ou sans cache

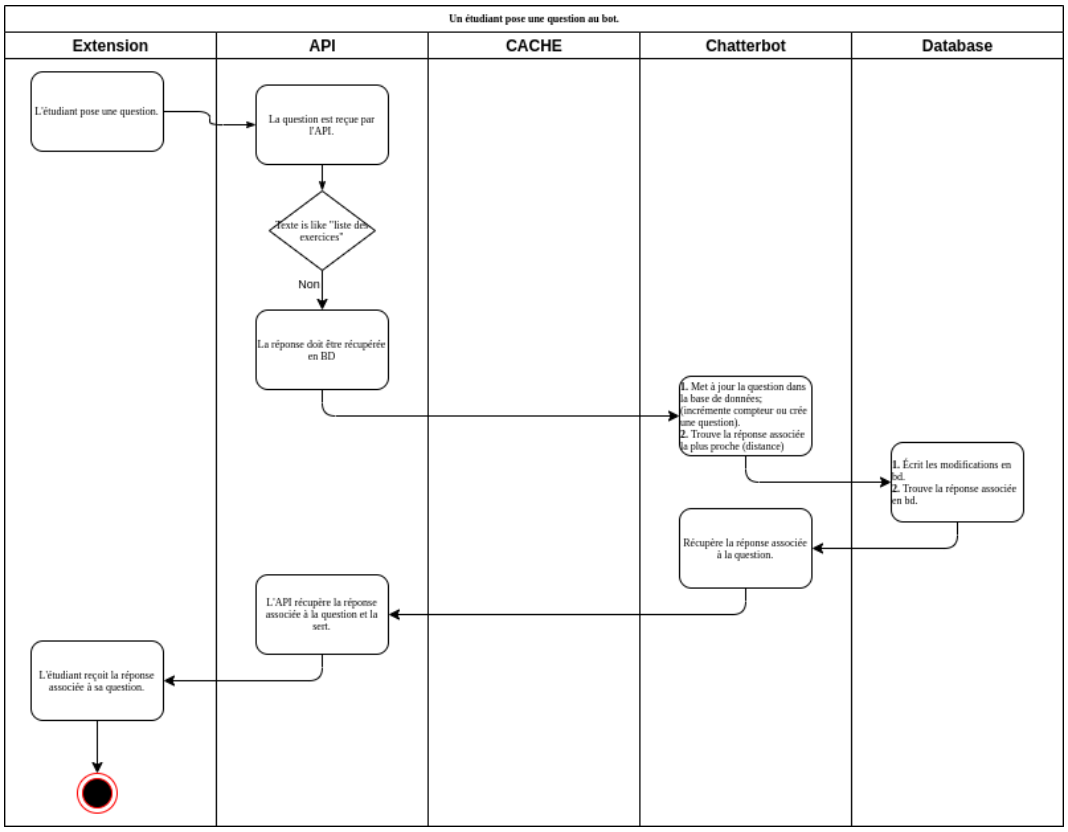


FIGURE 22 – Récupération d'une réponse à la suite d'une question posée au bot

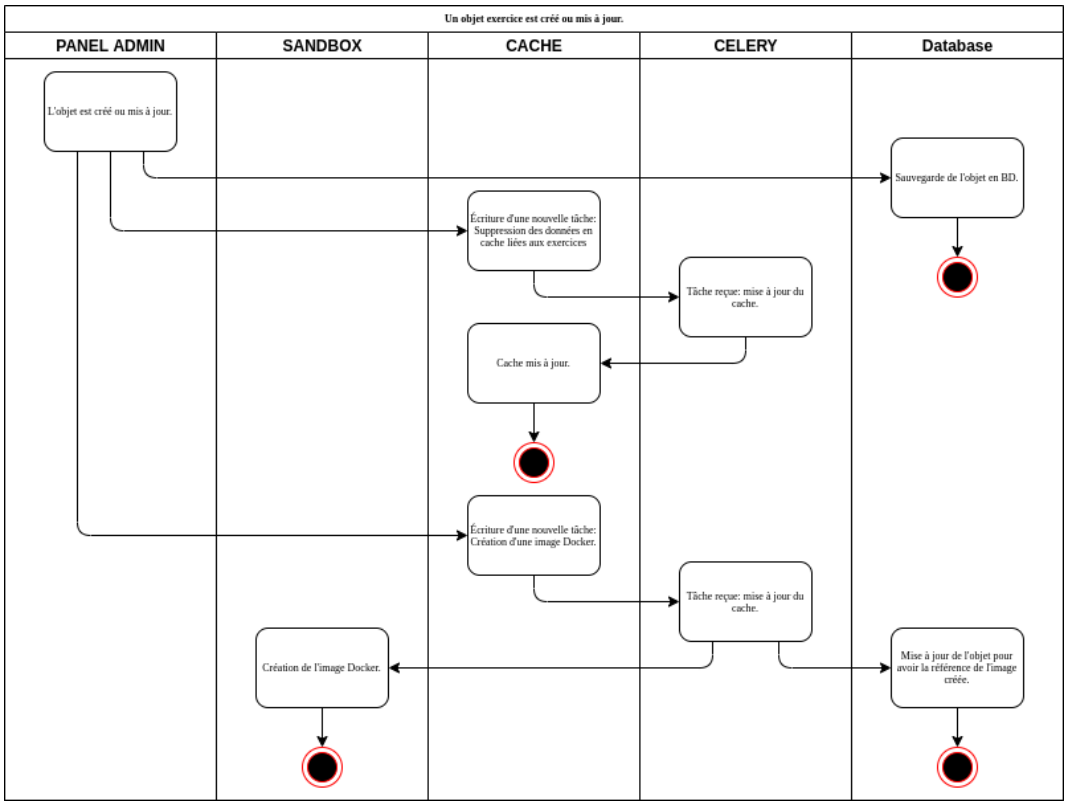


FIGURE 23 – Sauvegarde d'un exercice par un utilisateur-administrateur.

Toutes ces interactions sont également décrites dans la figure 23 et les différentes étapes de ce processus sont représentées par les actions en vert et celles en mauves. Les actions vertes étant celles exécutées directement et non déléguées au *Celery*.

Un utilisateur exécute un code

L'exécution de codes au sein de notre infrastructure est également une fonctionnalité clé.

Dans ce cas également, l'opération dure un certain temps. Cependant, comme lors d'une exécution normale, l'utilisateur « attend » le résultat de son exécution.

L'API va donc lancer un conteneur *Docker* correspondant à l'exercice résolu par l'utilisateur avec le code fourni. L'étudiant attendra la réponse de cette exécution. Si la réponse prend trop de temps à survenir, une erreur sera soulevée par le conteneur et un *timeout* sera renvoyé en tant que réponse. Une fois l'exécution réalisée, le résultat est envoyé sans attendre.

De plus, un système d'historique des exécutions est mis en place. Toutes celles-ci sont sauvegardées dans la BD et cette tâche est déléguée à un travailleur *Celery*, car elle n'est pas directement liée à l'exécution du code. Ces données seront potentiellement consultées plus tard par un professeur ou un assistant. Il n'est donc pas nécessaire que l'utilisateur « attende » ces écritures en BD.

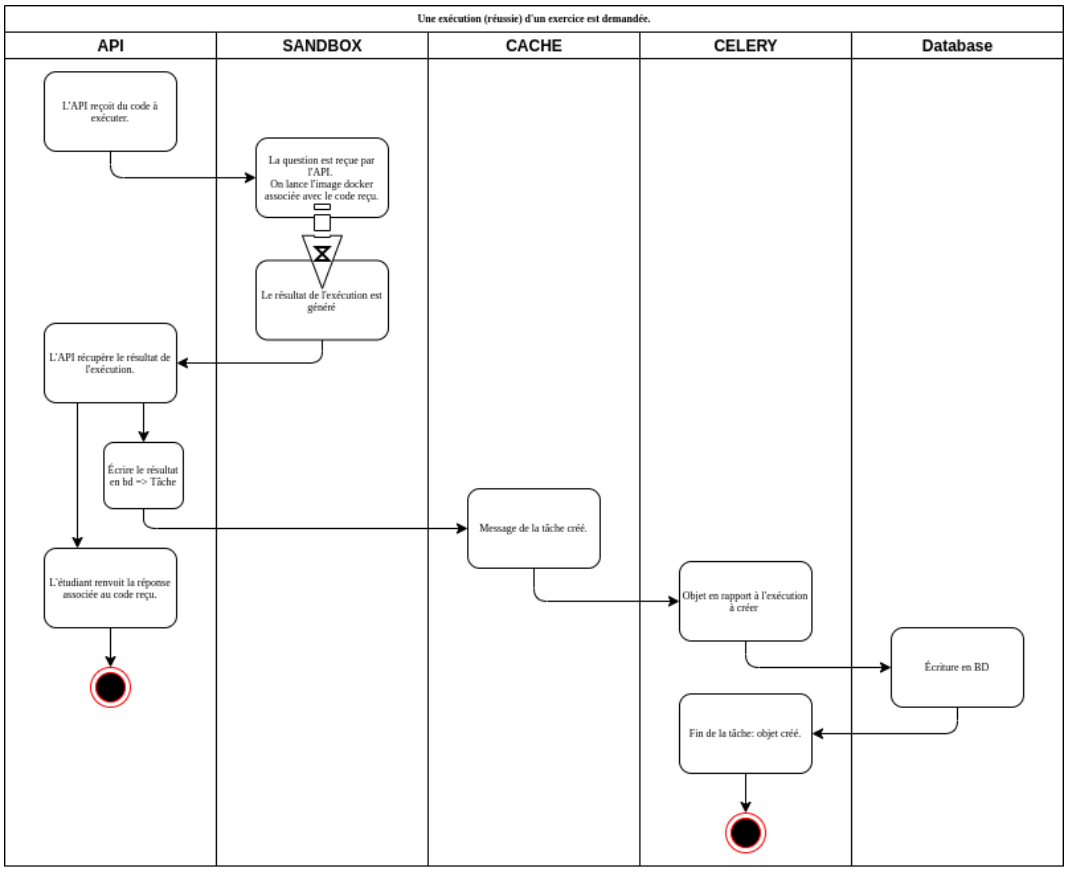


FIGURE 24 – Exécution d'un exercice fourni par un utilisateur.

Les modules impliqués pour ce type d'opération sont à nouveau repris dans la figure 24 en annexe.

En Bref ! : Interactions de nos composants

L'architecture proposée repose sur un ensemble de composants qui communiquent entre eux. Cette section permet de mieux comprendre ce qu'il se cache derrière une simple action du côté de l'utilisateur. Divers cas de figure ont été explorés afin d'expliciter ces interactions entre les composants.

3.3.5 Nom de la solution

Le nom donné pour la solution est BOTLEARN. Ce nom fait référence à plusieurs éléments en anglais en lien avec l'environnement dans lequel la solution a été développée.

- **bot** : la contraction de robot
- **learn** : qui signifie « apprendre » en anglais
- **butler** : qui signifie « majordome » en anglais

En Bref ! : Solution proposée

L'élément central de la solution, nommée BOTLEARN est un chatbot. Celui-ci a pour vocation de répondre aux questions des étudiants, mais également interpréter leur code. Ce chatbot repose sur un serveur qui a pour fonction d'exécuter le code des élèves, mais également de fournir un niveau d'abstraction suffisant pour faire fonctionner l'interface pédagogique. Cette interface est le lien entre les étudiants et le corps enseignant. En effet, celle-ci permet également le paramétrage du chatbot et la consultation des codes des étudiants.

3.4 Discussion

Quelques points spécifiques au projet doivent être abordés et de même quelques limites doivent être annoncées afin d'entrevoir d'autres pistes de réflexion pour le futur.

3.4.1 Pertinence des feedbacks

Durant ce travail, la perception des feedbacks a été effectuée en tenant compte de la « forme » de ceux-ci en fonction des critères explicités au préalable. Toutefois, à l'avenir, les professeurs et les assistants seront certainement amenés à envisager le travail sur le « fond » lors de l'entraînement du chatbot. La persévérance de l'étudiant dépend de la réponse reçue à chaque question. L'évaluation de la pertinence des feedbacks du bot se fera au fur et à mesure.

En effet, après quelques semaines d'utilisation, il sera possible de déterminer le degré de satisfaction des étudiants qui ont obtenu une réponse par le chatbot au moyen d'un questionnaire d'évaluation.

3.4.2 Maintenance et pérennité

BOTLEARN est amené à évoluer avec le temps. Il est donc très important de garder une trace des décisions prises dans le cadre de l'implémentation de la solution. Une documentation pour le back-end et le front-end est disponible pour expliquer ces choix. Le code est commenté afin de le comprendre rapidement. L'idée est d'avoir une documentation expliquant les grandes étapes et les points *sensibles* du projet tandis que les commentaires sont là pour comprendre plus en profondeur le code en lui-même.

3.4.3 Limites

Il semble logique d'avoir rencontré certaines limites en mélangeant le monde pédagogique et le monde professionnel.

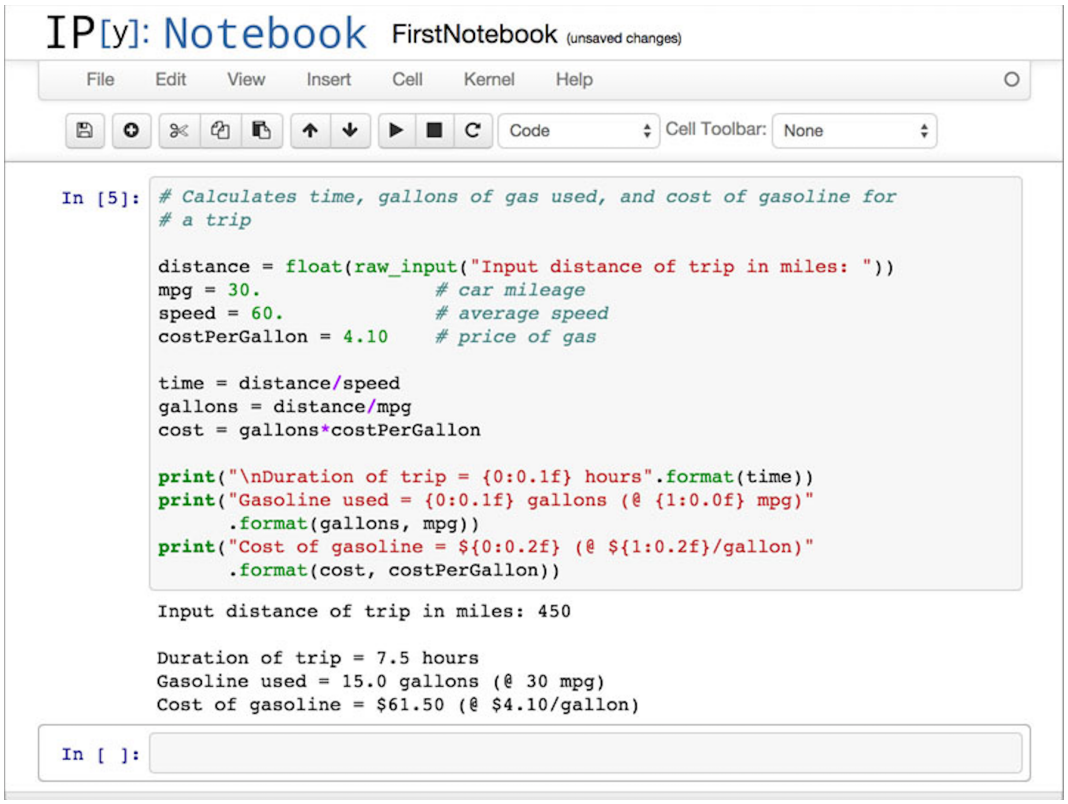


FIGURE 25 – Interface d'un notebook[38]

- Une première limite concerne l'accompagnement pas-à-pas d'une solution plus pédagogique. En prenant comme exemple les notebooks de *Canopy*, il était relativement facile pour les étudiants de coder grâce à son interface ludique. Le découpage de code était facile et le résultat de l'exécution était conservé (figure 25). Aussi, une transition de VSCode vers *Canopy* peut provoquer une charge cognitive plus élevée. Toutefois, comme il a été discuté auparavant, VSCode reste relativement clair dans son interface. C'est au niveau du démarrage que cela peut prendre plus de temps. L'avantage de VSCode permettra une plus grande facilité pour le passage vers de plus gros projets, car dans ce cas il faudra utiliser un éditeur de code et non plus un notebook (notamment le passage du cours *[INFOB131] Introduction à la programmation* à *[INFOB132] Projet de programmation*).

- Le chatbot en lui-même a des limites. En effet, même s'il est en mesure de comprendre des questions de programmation, le chatbot risque d'être incapable d'interpréter les questions posées si la question est imprécise en raison des lacunes des étudiants. De même, si l'étudiant pose des questions précises sur un exercice (tel qu'un problème de compréhension de l'exercice), le bot n'est actuellement pas capable d'y répondre.

- En l'état actuel, le résultat de l'exécution du code n'est pas examiné. Seule la sortie du code est effectivement examinée (à savoir s'il n'y a pas d'erreurs). Par exemple, que la réponse d'un étudiant à un exercice soit

38. https://physics.nyu.edu/pine/pymanual/html/apdx2/apdx2_ipynb.html

"*42*" ou "*Hello World*" n'a pas d'importance pour l'analyse. Le but est uniquement de vérifier « comment » l'étudiant est arrivé au résultat final, mais la finalité en elle-même n'est pas analysée. Il s'agit d'un choix, le but de ce projet n'était pas de faire une auto vérification de résultats puisqu'il existe des solutions prévues à cet effet.

- Le profil de l'étudiant n'est pas vraiment pris en compte avec BOTLEARN. En effet, les styles d'apprentissage ont tendance à être débattus dans la doctrine scientifique au niveau de sa pertinence. Il n'a donc pas semblé pertinent de catégoriser les étudiants selon leur style d'apprentissage. Par ailleurs, une autre raison de ce rejet est que l'exercice de catégorisation est complexe et nécessite d'être réalisé par un psychologue.

Comme indiqué, c'est avec le temps qu'il sera possible d'adapter ou améliorer le projet pour tendre vers une solution la plus adéquate pour les étudiants en apprentissage d'un langage de programmation.

En Bref ! : Discussion

La solution possède quelques points d'attention comme la gestion des feedbacks ainsi que la maintenabilité de BOTLEARN dans le temps. Le projet comporte également certaines limitations, celles-ci étant une base pour penser à de nouvelles fonctionnalités pour le futur. Ces nouvelles fonctionnalités seront abordées dans la section 5.

Résumé du chapitre ▶ Conception

Dans ce chapitre, la réalisation de notre projet ainsi que ses caractéristiques ont été abordées. Il est important de retenir qu'il est complexe d'apprendre la programmation. Pour surpasser ces difficultés, il est nécessaire de faire les bons choix (comme l'IDE, le langage de programmation, etc.).
Le but de notre solution est d'arriver à procurer un sentiment de confort lorsqu'un étudiant rencontre un problème. En temps normal lorsque celui-ci est confronté à une difficulté, il est amené à poser des questions aux professeurs et assistants. C'est en obtenant une réponse rapide qui correspond à sa question que celui-ci est satisfait. C'est pour cette raison que le choix du chatbot a été sélectionné. Celui-ci peut simuler (en partie) l'interaction humaine en répondant à des questions « simples ».
L'infrastructure de la solution est très importante pour sa

flexibilité et sa maintenance, chaque composant du projet communique avec les autres afin d'obtenir une homogénéité des services.

4 Validation

Dans ce chapitre, les tests utilisateurs réalisés sur un panel de personnes (à savoir les professeurs et assistants) seront détaillés grâce à un questionnaire sur la forme de la solution. Le but était d'apprécier la clarté, la prise en main et la compréhension de la solution.

Tester le fond de la solution est beaucoup plus compliqué surtout au niveau des étudiants. BOTLEARN a été créé afin de répondre aux questions d'étudiants peu aguerris à la programmation. Dès lors, réaliser des tests fin avril sur les étudiants de Bloc 1 n'aurait amené que très peu de valeur ajoutée puisque leurs interrogations ont évolué au fil de l'année en même temps que leurs connaissances se sont agrandies (suite aux cours et différents projets). Tout ceci a pu être constaté grâce au HelpDesk qui avait été prévu pour le projet de programmation. La solution proposée doit être validée en grosse partie en situation réelle. C'est-à-dire dès le début du quadrimestre 1. C'est à ce moment-là que le chatbot sera normalement en adéquation avec les attentes des étudiants.

4.1 Méthodologie

Pour ces tests nous avons utilisé le User Experience Questionnaire (UEQ) [39] afin d'obtenir une impression globale de l'expérience de l'utilisateur. Ce questionnaire permet de mesurer les aspects de la convivialité (efficacité, perspicacité, fiabilité), les aspects de l'expérience de l'utilisateur (originalité, stimulation) et l'aspect concernant l'attractivité.

Plus précisément, le questionnaire permet de juger un projet sur six aspects [Schrepp et al., 2017].

- attractivité : Impression générale du produit. Les utilisateurs aiment-ils ou n'aiment-ils pas le produit ?
- perspicacité : Est-il facile de se familiariser avec le produit ? Est-il facile d'apprendre à utiliser le produit ?
- efficacité : Les utilisateurs peuvent-ils résoudre leurs tâches sans effort inutile ?
- fiabilité : L'utilisateur a-t-il le sentiment de maîtriser l'interaction ?
- stimulation : L'utilisation du produit est-elle excitante et motivante ?
- originalité : le produit est-il innovant et créatif ? Le produit suscite-t-il l'intérêt des utilisateurs ?

Pour l'attractivité, il y a six questions, tandis que toutes les autres questions en comportent quatre. La figure 26 montre la structure du questionnaire.

39. https://www.ueq-online.org

	1	2	3	4	5	6	7		
Agaçant	○	○	○	○	○	○	○	Agréable	1
Incompréhensible	○	○	○	○	○	○	○	Compréhensible	2
Moderne	○	○	○	○	○	○	○	Sans fantaisie	3
Appropriation simple	○	○	○	○	○	○	○	Appropriation compliquée	4
Apporte de la valeur	○	○	○	○	○	○	○	Peu de valeur ajoutée	5
Ennuyeux	○	○	○	○	○	○	○	Captivant	6
Inintéressant	○	○	○	○	○	○	○	Intéressant	7
Imprévisible	○	○	○	○	○	○	○	Prévisible	8
Rapide	○	○	○	○	○	○	○	Lent	9
Original	○	○	○	○	○	○	○	Conventionnel	10
Rigide	○	○	○	○	○	○	○	Facilitant	11
Bien	○	○	○	○	○	○	○	Médiocre	12
Compliqué	○	○	○	○	○	○	○	Simple	13
Repoussant	○	○	○	○	○	○	○	Attractif	14
Habituel	○	○	○	○	○	○	○	Avant-gardiste	15
Désagréable	○	○	○	○	○	○	○	Agréable	16
Sécurisant	○	○	○	○	○	○	○	Insécurisant	17
Stimulant	○	○	○	○	○	○	○	Soporifique	18
Répond aux attentes	○	○	○	○	○	○	○	Ne répond pas aux attentes	19
Inefficace	○	○	○	○	○	○	○	Efficace	20
Clair	○	○	○	○	○	○	○	Déroutant	21
Non pragmatique	○	○	○	○	○	○	○	Pragmatique	22
Sobre	○	○	○	○	○	○	○	Surchargé	23
Attrayant	○	○	○	○	○	○	○	Rébarbatif	24
Sympathique	○	○	○	○	○	○	○	Inamical	25
Conservateur	○	○	○	○	○	○	○	Innovant	26

FIGURE 26 – Questionnaire en 26 questions.

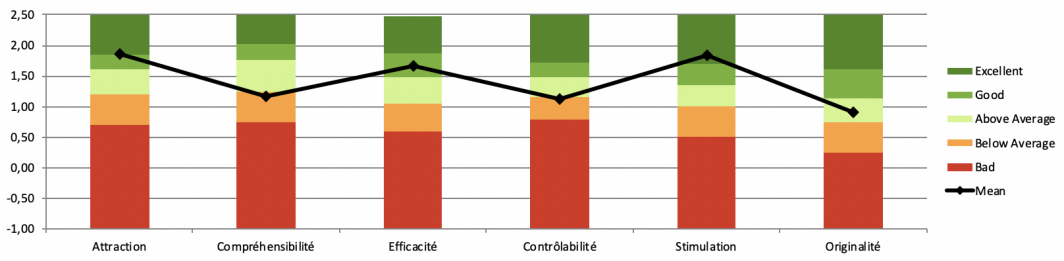


FIGURE 27 – Comparaison de BOTLEARN par rapport à 452 autres produits informatiques.

Une fois le questionnaire rempli, il est possible d'interpréter les données récoltées à travers des graphes mis à disposition par cette méthodologie. Nous y reviendrons dans la section 4.2 parlant des résultats.

4.2 Résultats

Sur base des résultats obtenus, il est possible de dégager certaines tendances et certains points d'attention. Néanmoins, il est important de nuancer ces résultats obtenus auprès des utilisateurs (professeurs et assistants) étant donné que l'échantillon est de six personnes. Les résultats pourraient donc varier avec le test de personnes supplémentaires.

La figure 27 permet de comparer la solution informatique par rapport à 452 autres. En effet, dans les outils d'analyse que propose ce questionnaire, celui-ci donne la possibilité de situer sa solution par rapport à 452 autres évaluations de produits informatiques par un graphe. Il est possible de mettre en perspective sa solution par rapport à d'autres. En effet, le formulaire se base essentiellement sur la forme et non sur le fond. Il est donc concevable d'effectuer une comparaison de la solution proposée par rapport à d'autres existantes. Selon la situation du point noir, il est possible de savoir si la solution se trouve dans la moyenne, en dessous ou au-dessus.

Il en ressort que BOTLEARN se démarque par son attractivité et sa stimulation. Par contre, la solution semble moins compréhensible et contrôlable que d'autres. Cela peut s'expliquer par la présentation de l'information, mais aussi au niveau de ses fonctionnalités. Il était très important de mettre en avant les fonctionnalités intéressantes afin de montrer le potentiel de la solution, mais ces fonctionnalités n'étant pas toutes simples à prendre en main ou simplement à comprendre, cela peut requérir un temps d'adaptation. Il nous a d'ailleurs été rapporté plusieurs fois pendant les interviews que la solution une fois expliquée paraissait plus simple. Pour revenir sur le côté stimulant, il semble que la promesse d'une solution engendre une certaine attente du côté du corps enseignant. Proposer un outil destiné à accompagner le quotidien de l'étudiant et de l'enseignant suscite la curiosité et l'envie de découvrir.

Concernant l'originalité, nous nous trouvons légèrement au-dessus de la moyenne. Cela pourrait être dû au fait que notre projet est une solution Web. Ce point est relativement complexe à détailler, car la solution en elle-même

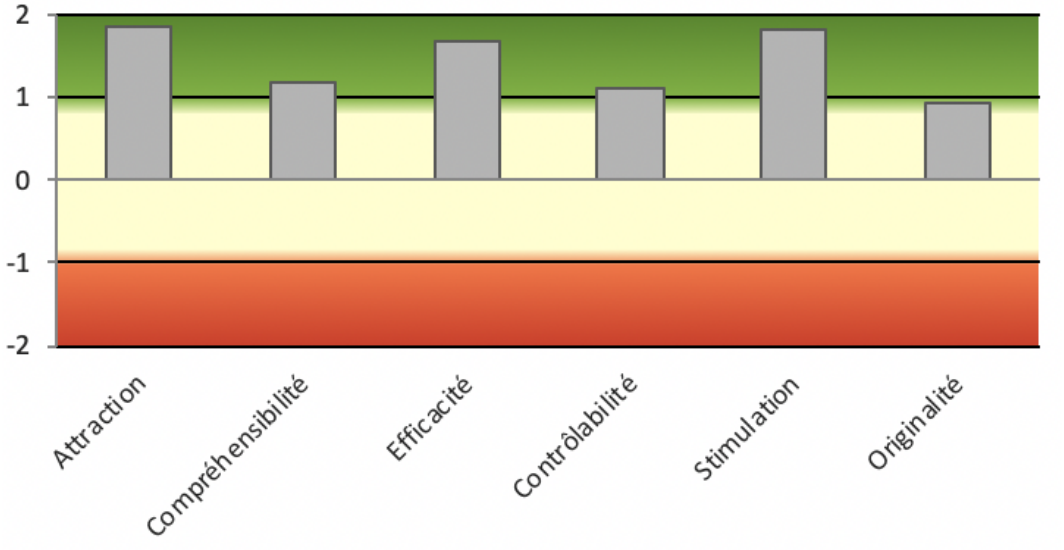


FIGURE 28 – Résultats bruts des questionnaires.

semble relativement innovante. À l'heure actuelle, il ne semble pas exister de solution comparable à celle proposée dans le cadre de ce mémoire. Toutefois lorsque l'on considère uniquement la forme, cela reste une plateforme Web. C'est avec ce genre de cas que nous pouvons constater les limites de l'échantillon. Il n'est pas toujours aisé d'interpréter les résultats.

Pour terminer, il reste l'efficacité. BOTLEARN semble efficace dans le sens où les utilisateurs ayant voulu réaliser une tâche y sont arrivés de manière rapide. Cette partie est liée à la compréhension de la solution. Il est vrai qu'il a fallu un temps d'adaptation aux utilisateurs, mais une fois compris, ceux-ci ont pu réaliser les tâches demandées sans trop de problèmes. Cela permet de démontrer en partie que la solution est pragmatique dans sa conception.

Considérons les résultats bruts, sans comparaison avec d'autres produits avec la figure 28. L'interprétation est relativement similaire par rapport à celle réalisée précédemment ; on note tout au plus une petite augmentation pour le côté compréhensible de la solution. Malgré le fait que la solution soit légèrement moins compréhensible que la moyenne, celle-ci reste tout de même accessible. L'intérêt de ce graphe permet d'obtenir une vision d'ensemble d'une solution informatique sur base de six critères et permet donc d'identifier les points d'attention. Les figures 27 et 28 permettent d'être complémentaires afin d'améliorer la solution proposée. D'autres graphes plus détaillés se trouvent dans les annexes.

Après l'analyse des résultats produits par le questionnaire, des questions orales ont été soumises aux personnes qui ont testé BOTLEARN :

- Comment s'est passée l'utilisation du site de manière générale ? Quel est le ressenti ?
- Est-ce que de prime abord la solution pourrait aider dans le quotidien en sachant que celle-ci pourra être améliorée dans le futur ?

- Quelle est la fonctionnalité la plus utile après utilisation (du point de vue de l'administrateur du site et éventuellement du côté étudiant) ?
- Quelques améliorations ou quels ajouts peuvent contribuer à améliorer la solution ?

Le but de ces questions était de cibler certains points très importants de BOTLEARN. Il est fortement probable que le projet en lui-même soit amené à évoluer dans le temps. Il était donc extrêmement important d'identifier les points forts et les potentielles idées futures afin de consolider ce projet sur base des points forts cités.

Les échanges avec les différentes personnes se sont avérés très enrichissants. Il est régulièrement ressorti des échanges le besoin d'intégrer l'exécution dans la solution. Le fait que le code ne soit pas exécuté sur la machine des étudiants évite de nombreux problèmes liés à la version du code et fait gagner un temps considérable aux assistants en leur évitant de venir aider pour des problèmes liés à l'installation de Python. Au début, l'étudiant peut se familiariser avec le langage Python et ses préoccupations sont uniquement d'ordre algorithmique ou syntaxique. Un autre point systématiquement abordé concerne les statistiques. Pouvoir obtenir des informations sur les difficultés rencontrées par les étudiants en un coup d'oeil semble être considéré comme une plus-value importante dans le quotidien des professeurs et assistants.

Un autre point qui a été soulevé concerne la possibilité de paramétrer le chatbot. Pouvoir définir les réponses données par le chatbot permet de rester dans les limites du cours. Si l'étudiant venait à poser une question sur internet, il pourrait obtenir énormément de réponses différentes dont certaines pourraient être mauvaises, hors propos ou simplement trop complexes. En créant les réponses, on assure que l'étudiant recevra uniquement l'information nécessaire à sa question et rien de plus.

Concernant les modifications et ajouts qui ont été discutés, ceux-ci se retrouvent dans la section 5.

Résumé du chapitre ► Validation

Dans ce chapitre, la solution a été confrontée aux acteurs qui seront amenés à faire vivre le projet. Il a été démontré que cette solution semble répondre à un grand nombre d'attentes et peut aider au quotidien les professeurs et assistants. L'échantillon étant relativement restreint, l'analyse des résultats permet de dégager des tendances, mais tirer des conclusions de manière définitive apparaît prématuré.
La grande inconnue reste l'utilisation que les étudiants pourront en faire. Même si l'utilisation du HelpDesk (discuté dans la

section 3.1.1) a permis l'identification des besoins principaux des étudiants ainsi que la connaissance de leurs questions lors de leur découverte de la programmation, il n'est pas garanti que le chatbot dans sa forme actuelle leur apporte la meilleure solution. Il faudra donc tester le logiciel en situation réelle afin de vérifier son utilisation et proposer des ajustements au besoin.

5 Travaux futurs

Dès la conception du projet BOTLEARN, il est rapidement apparu que des limites devaient être définies dans sa description initiale et que le système devait évoluer et être amélioré, principalement par l'enrichissement des retours effectués par les premiers utilisateurs, étudiants et enseignants.

Il s'agit d'une première version et plusieurs pistes d'amélioration peuvent être définies.

5.1 Enrichissement de la base de données

L'objectif essentiel de l'application est la capacité à fournir une réponse aux étudiants, à la fois pertinente, mais également en adéquation avec leurs attentes. C'est l'évaluation continue de la qualité des réponses proposées et surtout de la satisfaction des utilisateurs qui permettra de garantir la poursuite de l'utilisation du système d'aide et d'éviter un abandon lié à la lassitude d'étudiants insatisfaits.

L'enrichissement continuel de la base de données doit fournir une meilleure garantie quant au maintien de l'efficacité du système, principalement par l'ajout de questions/réponses qui permettent de couvrir l'ensemble des questions posées par les étudiants.

L'analyse des données récoltées par les premiers utilisateurs devra permettre de nettoyer certaines imperfections et d'agréger de nouvelles informations qui seront continuellement ajoutées à la base de données et permettront ainsi une amélioration progressive de l'offre proposée par l'application.

5.2 Analyse du code

Pour l'instant, l'analyse de code fonctionne sur base de code d'erreurs (ou "warning") disponibles grâce à *Flake8*[40]. De ce fait, toutes les erreurs imaginables ne sont pas traitées. Toutefois, il est possible et assez simple, d'installer des plug-ins *Flake8* qui couvrent d'autres erreurs ou qui vérifient les bonnes pratiques. Par exemple, il existe des plug-ins permettant de vérifier le nom des variables, afin de s'assurer que l'étudiant utilise un nom adéquat. La communauté a réalisé beaucoup de plug-ins différents[41] mais rien n'empêche de développer un plug-in « maison »[42].

Chacun de ces plug-ins définit une série de codes d'erreurs. Ces codes sont rédigés en anglais et peuvent donc nécessiter une traduction pour une meilleure intégration dans le système.

Cependant, tous ces codes d'erreurs ne sont pas toujours essentiels pour des développeurs débutants. Ils peuvent être désactivés nativement, ou peuvent

40. https://flake8.pycqa.org/en/latest/
41. https://github.com/DmytroLitvinov/awesome-flake8-extensions
42. https://flake8.pycqa.org/en/latest/plugin-development/index.html

même être activés ou désactivés en fonction de l'état de progression des étudiants. Par exemple, lors des premiers TP, les règles liées au nommage des variables pourraient être désactivées et ne redeviendraient actives que lorsque les enseignants considèrent que le concept est maitrisé par la majorité des étudiants. L'intérêt reste le contrôle de la charge cognitive des étudiants et ainsi éviter une surcharge qui peut leur être délétère. Une manière d'implémenter cette fonctionnalité serait de fonctionner avec des "templates" regroupant les codes d'erreurs. Ceux-ci peuvent également être regroupés en ensembles définis (groupes) et être ainsi assignés de façon spécifique à certains exercices précis.

Le choix de travailler avec *Flake8* s'est imposé pour sa simplicité et la haute configuration possible. Cependant, il est tout à fait envisageable de changer d'outil d'analyse de code statique(p. ex., *sonarqube*[43]). Les changements sur l'architecture sont envisageables puisque l'analyse des codes est bien isolée dans un conteneur *Docker*.

Le chatbot propose un feedback sur la façon de coder et non sur le résultat attendu de l'exercice. Il pourrait s'avérer intéressant qu'à terme, le chatbot puisse également vérifier si la réponse de l'étudiant est celle qui est attendue pour l'exercice. Avoir un feedback personnalisé sur ces deux points pourrait s'avérer plus intéressant pour l'étudiant, mais la pertinence de cette approche doit, au préalable, être étudiée et confirmée par de nouvelles recherches.

5.3 Statistiques

Les professeurs et assistants disposent sur le Web d'une section « *statistiques* ». De toute évidence, la pertinence et la précision de ces données statistiques vont dépendre de la quantité d'informations qui pourra être extraite, ainsi que du développement continuel de l'outil, comme discuté auparavant.

Un autre point d'intérêt réside dans la possibilité d'identification de la ligne de codage à la source de l'erreur. Actuellement, il est possible de déterminer les erreurs d'un code réalisé par l'étudiant, mais la ligne précise erronée n'est pas signalée. Identifier de façon spécifique cette ligne n'est pas difficile par le système, mais peut être fort utile pour les enseignants-administrateurs de la plateforme : une aide correctrice est beaucoup plus facile et rapide et ceci permet donc, en fin de compte, d'optimaliser de façon significative le temps d'enseignement alloué à chaque étudiant.

Un autre apport *statistique* pourrait être fourni aux étudiants eux-mêmes. Ceux-ci pourraient se connecter sur la plateforme Web, sur une section précise et avec leur identifiant propre, et ainsi visualiser rapidement la situation de leur apprentissage et surtout leur niveau de progression.

5.4 Gamification

La gamification est un moyen d'apprentissage reconnu dans l'informatique. Gamifier a déjà fait ses preuves dans le domaine de l'apprentissage de l'infor-

43. https://www.sonarqube.org/

matique. En effet, il est relativement courant de voir que l'apprentissage de la programmation se fait à travers un jeu. Il est possible de citer *Scratch*[44] ou dans un autre style *CodInGame*[45]. Quel que soit le public visé, le principe reste le même. Il s'agit de rajouter un aspect ludique à l'apprentissage de la programmation pour, par exemple, simplifier des concepts ou encore motiver les étudiants.

Le but ici ne serait pas de créer un jeu, mais il pourrait être intéressant d'aller plus loin vis-à-vis de la stimulation des étudiants à résoudre des exercices. Il est difficile de juger à priori si l'étudiant souhaitera résoudre les exercices proposés par le chatbot. Il s'agit d'une toute nouvelle façon de faire. Toutefois, la gamification pourrait passer par une barre de progression dans les exercices ou encore afficher son taux d'erreur d'une manière à ce que l'étudiant se motive à s'améliorer davantage. Les personnes visées par le chatbot restent un public débutant qui découvre l'univers de la programmation. Il est important de garder la tâche de programmation telle quelle pour que les étudiants apprennent la programmation dans son ensemble, mais ajouter sur le côté des aspects de gamification pourrait améliorer l'expérience utilisateur.

5.5 Amélioration des Sandboxes

Botlearn utilise des sandboxes afin d'exécuter les codes des étudiants. Cette solution est couteuse en ressources et en temps, mais semble être l'unique solution réaliste. D'autres solutions pourraient exister pour l'exécution de codes étrangers de manière sécurisée (p. ex. virtualisation complète avec un hyperviseur).

Il est possible d'aller encore plus loin grâce à *PYPY*[46]. Ce dernier est un interpréteur Python alternatif écrit en Python ; à distinguer donc de l'interpréteur classique de Python, appelé *CPython*, écrit en C.

Il existe différents avantages à utiliser *PYPY* à la place de *CPython* :

- Les fonctionnalités de "sanboxing" sont déjà prévues et implémentées d'une manière portative[47]. Rajouter cette seconde couche d'encapsulation ne semble pas inutile. En effet, la solution fournit déjà un contrôle sur les ressources qui peuvent être utilisées, mais avec ce sandbox, il pourrait être rajouté un contrôle sur les commandes potentiellement dangereuses et ainsi pouvoir les exclure de l'exécution.

- De plus, les développeurs de *PYPY* signalent que leur interpréteur peut être sensiblement plus rapide[48] que celui par défaut de Python. Ils mettent à disposition des graphiques qui comparent les performances. Il est cependant important de signaler l'absence de ces gains de performance lorsque

44. http://scratchfr.free.fr
45. https://www.codingame.com/home
46. https://doc.pypy.org/en/latest/index.html
47. https://doc.pypy.org/en/latest/sandbox.html#pypy-s-sandboxing-features
48. https://speed.pypy.org/

le programme Python utilise des librairies écrites en C (p. ex. *Numpy*[49]. En effet, lorsque *PYPY* doit utiliser ce genre de librairie, l'interpréteur doit émuler la manière dont les objets sont stockés en mémoire ce qui n'est pas efficace. À titre informatif, l'équipe de développement de *PYPY* travaille également sur une nouvelle API[50] qui pourrait solutionner ce défaut dans le futur.

Avant d'appliquer ce changement, il sera important de considérer les différences[51] qui existent entre les deux interpréteurs et déterminer si celles-ci peuvent avoir un impact dans le cadre d'exécution de codes simples. Le but n'est pas d'avoir des différences visibles dans les exécutions entre ces deux solutions.

Il existe une image *Docker* officielle[52] pour cet interpréteur. De ce fait, passer à cet interpréteur alternatif nécessiterait de régénérer tous les exercices sur base de cette image.

5.6 Poursuivre la validation

La solution développée tout au long de ce mémoire a pu être validée auprès du corps enseignant. Cette validation a permis de juger le produit sur six aspects : l'attractivité, la perspicacité, la fiabilité, la stimulation et l'originalité. Grâce au résultat de ces tests, BOTLEARN a pu être amélioré afin de répondre au mieux à certaines attentes des professeurs et assistants.

Toutefois, ce projet s'adresse en premier lieu aux étudiants qui apprennent la programmation. Pour les raisons indiquées dans le chapitre 4 - Validation, il n'a pas été possible de faire valider la solution par les premiers intéressés. Dès le début d'année universitaire 2020 et en parallèle avec le cours *[INFOB131] Introduction à la programmation*, le chatbot devra être éprouvé par les étudiants. Pour réaliser ces tests correctement, une méthodologie en deux phases pourrait être mise en place comme suit :

- Premièrement, afin d'être complémentaire avec la validation « partielle » qui a été mise en place (chapitre 4), il faudrait soumettre ce même questionnaire à quelques élèves. L'objectif serait de juger la forme du chatbot et d'améliorer l'interface afin de rendre optimale son utilisation.
- Deuxièmement, la mise en place d'un questionnaire à destination de tous les étudiants-utilisateurs semble primordiale. Celui devra permettre de jauger l'intérêt (ou non) du chatbot selon eux, mais également la qualité et la pertinence des réponses. L'utilisation du chatbot par les étudiants devra également être mesurée. Toutes ces informations récoltées pourront être mises en perspective avec les données produites par le système.

49. https://numpy.org/
50. https://github.com/pyhandle/hpy
51. https://doc.pypy.org/en/latest/cpython_differences.html
52. https://hub.docker.com/_/pypy

Utiliser ces deux approches semble pertinent afin de réaliser une validation complète du chatbot par les étudiants. C'est grâce à cette double validation (autant pour le corps enseignant que pour les étudiants) que la pertinence et l'efficacité de BOTLEARN pourront être évaluées.

Résumé du chapitre ► Travaux futurs

Dans ce chapitre, des pistes de réflexion ont été envisagées afin de poursuivre le travail du mémoire et du stage. Ces changements concernent autant des améliorations au niveau de la qualité de ce que fournit le chatbot (et donc observable par les utilisateurs) que des améliorations au niveau du fonctionnement interne de BOTLEARN. Celles-ci devront être mises en perspective avec une nouvelle validation de la solution.
De nouveaux aspects pédagogiques peuvent également être ajoutés à la solution. La gamification de l'apprentissage est une des pistes proposées.

6 Conclusion

Apprendre à programmer est un challenge considérable, difficile, exigeant. Un temps important est nécessaire pour obtenir une maîtrise parfaite d'un langage de programmation. Les débuts sont ardus et pourtant il est essentiel de bien connaître les bases, de les assimiler, avant de pouvoir comprendre les notions suivantes plus complexes, et ainsi pouvoir progresser dans la programmation, tout en visualisant correctement les écueils et en évitant la répétition des erreurs.

La pédagogie de l'apprentissage de la programmation a été étudiée et les facteurs déterminants à la réussite ont été identifiés, principalement cognitifs et psychologiques.

De nombreuses solutions sont proposées pour aider l'étudiant dans cet apprentissage difficile, mais une assistance reste fondamentale. Et il est souvent difficile d'offrir aux étudiants, toujours plus nombreux, un temps d'encadrement suffisant et adapté à leurs besoins et attentes.

Dans ce travail, un nouvel outil est proposé, BOTLEARN, de type conversationnel, axé principalement sur la qualité du code et attentif à optimaliser tous les facteurs d'apprentissage.

L'étudiant peut ainsi disposer d'une aide personnalisée au moyen d'un outil avec lequel il peut interagir en français. Il reçoit ainsi une aide efficace et motivante, qui l'aide à progresser dans sa capacité à produire un code de qualité, sans devoir se concentrer sur l'installation de librairie ou la configuration du glslangage_programmation.

BOTLEARN fournit également un support pédagogique aux enseignants en leur livrant des informations sur l'évolution de l'apprentissage de leurs étudiants, sur les questions et difficultés récurrentes à la fois pour l'ensemble d'un groupe, mais aussi sur chacun d'eux individuellement.

BOTLEARN est un outil qui vise à la fois à s'adapter aux exigences des étudiants, mais aussi à répondre aux défis de l'enseignement. Son design pratique et efficace ainsi que son potentiel d'amélioration future peuvent faire de BOTLEARN un outil indispensable dans l'apprentissage difficile de la programmation informatique.

7 Bibliographie

Références

[Abu Shawar and Atwell, 2007] Abu Shawar, B. and Atwell, E. (2007). Chatbots : are they really useful ? *LDV-Forum : Zeitschrift für Computerlinguistik und Sprachtechnologie*, 22(1) :29–49.

[Annamaa, 2015] Annamaa, A. (2015). Introducing thonny, a python ide for learning programming. *ACM International Conference Proceeding Series*, 19-22-Nov- :117–121.

[Benotti et al., 2014] Benotti, L., Martínez, M. C., and Schapachnik, F. (2014). Engaging high school students using Chatbots. *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference*, pages 63–68.

[Berglund and Lister, 2010] Berglund, A. and Lister, R. (2010). Introductory Programming and the Didactic Triangle. *Conferences in Research and Practice in Information Technology Series*, 103(Ace) :35–44.

[Bosch et al., 2013] Bosch, N., D'Mello, S., and Mills, C. (2013). What Emotions Do Novices Experience during Their First Computer Programming Learning Session ? *AIED 2013 : Artificial Intelligence in Education*, pages 11–20.

[Carmo et al., 2006] Carmo, L., Gomes, A., Pereira, F., and Mendes, A. (2006). Learning styles and problem solving strategies. In *Proceedings of 3rd E-Learning Conference–Computer Science Education, Coimbra, Portugal*, pages 7 – 8. September.

[Cherenkova et al., 2014] Cherenkova, Y., Zingaro, D., and Petersen, A. (2014). Identifying challenging CS1 concepts in a large problem dataset. *SIGCSE 2014 - Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 695–700.

[Cox, 2019] Cox, G. (2019). ChatterBot. https://chatterbot.readthedocs.io/en/stable/training.html. Accédé le : 2020-02-17.

[Cutts et al., 2010] Cutts, Q., Cutts, E., Draper, S., O'Donnell, P., and Saffrey, P. (2010). Manipulating mindset to positively influence introductory programming performance. *SIGCSE'10 - Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, pages 431–435.

[Entwistle, 1998] Entwistle, N. (1998). Motivation and approaches to learning : Motivating and conceptions of teaching. In Brown & Saly (Educational Development Adviser at University of Northumbria), editor, *Motivating Students*, chapter Motivating, pages 15–24. Elsevier Ltd., London, kogan page edition.

[Felder and Soloman, 2011] Felder, R. M. and Soloman, B. A. (2011). Everybody is active sometimes and reflective sometimes. *Strategies*, pages 4–6.

[Jenkins, 2001] Jenkins, T. (2001). The motivation of students of programming. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, pages 53–56.

[Kinnunen and Simon, 2010] Kinnunen, P. and Simon, B. (2010). Experiencing programming assignments in CS1 : The emotional toll. *ICER'10 - Proceedings of the International Computing Education Research Workshop*, pages 77–85.

[Kinnunen and Simon, 2011] Kinnunen, P. and Simon, B. (2011). CS majors' self-efficacy perceptions in CS1 : Results in light of social cognitive theory. *ICER'11 - Proceedings of the ACM SIGCSE 2011 International Computing Education Research Workshop*, pages 19–26.

[Kölling et al., 2003] Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. (2003). The bluej system and its pedagogy. *International Journal of Phytoremediation*, 21(1) :249–268.

[Mason and Cooper, 2014] Mason, R. and Cooper, G. (2014). Introductory programming courses in Australia and New Zealand in 2013 - trends and reasons. *Conferences in Research and Practice in Information Technology Series*, 148 :139–147.

[Mason et al., 2015] Mason, R., Cooper, G., Simon, and Wilks, B. (2015). Using cognitive load theory to select an environment for teaching mobile apps development. In *Conferences in Research and Practice in Information Technology Series*, volume 160.

[Morrison et al., 2014] Morrison, B. B., Dorn, B., and Guzdial, M. (2014). Measuring cognitive load in introductory CS. *CER 2014 - Proceedings of the 10th Annual International Conference on International Computing Education Research*, pages 131–138.

[Murphy and Thomas, 2008] Murphy, L. and Thomas, L. (2008). Dangers of a fixed mindset : Implications of self-theories research for computer science education. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, pages 271–275.

[Qian and Lehman, 2017] Qian, Y. and Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming : A literature review. *ACM Transactions on Computing Education*, 18(1) :1–24.

[Robins et al., 2006] Robins, A., Haden, P., and Garner, S. (2006). Problem distributions in a CS1 course. *Conferences in Research and Practice in Information Technology Series*, 52 :165–173.

[Schrepp et al., 2017] Schrepp, M., Hinderks, A., and Thomaschewski, J. (2017). Construction of a Benchmark for the User Experience Questionnaire (UEQ). *International Journal of Interactive Multimedia and Artificial Intelligence*, 4(4) :40.

[Scott and Ghinea, 2014] Scott, M. J. and Ghinea, G. (2014). On the domain-specificity of mindsets : The relationship between aptitude beliefs and programming practice. *IEEE Transactions on Education*, 57(3) :169–174.

[Sorva and Sirkiä, 2010] Sorva, J. and Sirkiä, T. (2010). UUhistle - A software tool for visual program simulation. *Proceedings of the 10th Koli Calling*

International Conference on Computing Education Research, Koli Calling'10, pages 49–54.

[Tandon and Ravikumar, 2013] Tandon, R. and Ravikumar, P. (2013). On the difficulty of learning power law graphical models. *IEEE International Symposium on Information Theory - Proceedings*, pages 2493–2497.

[Trekhleb, 2018] Trekhleb, O. (2018). Dynamic Programming vs Divide-and-Conquer. https://itnext.io/dynamic-programming-vs-divide-and-conquer-2fea680becb. Accédé le : 2020-02-17.

[Zingaro et al., 2018] Zingaro, D., Taylor, C., Porter, L., Clancy, M., Lee, C., Liao, S. N., and Webb, K. C. (2018). Identifying student difficulties with basic data structures. In *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 169–177.

8 Annexes

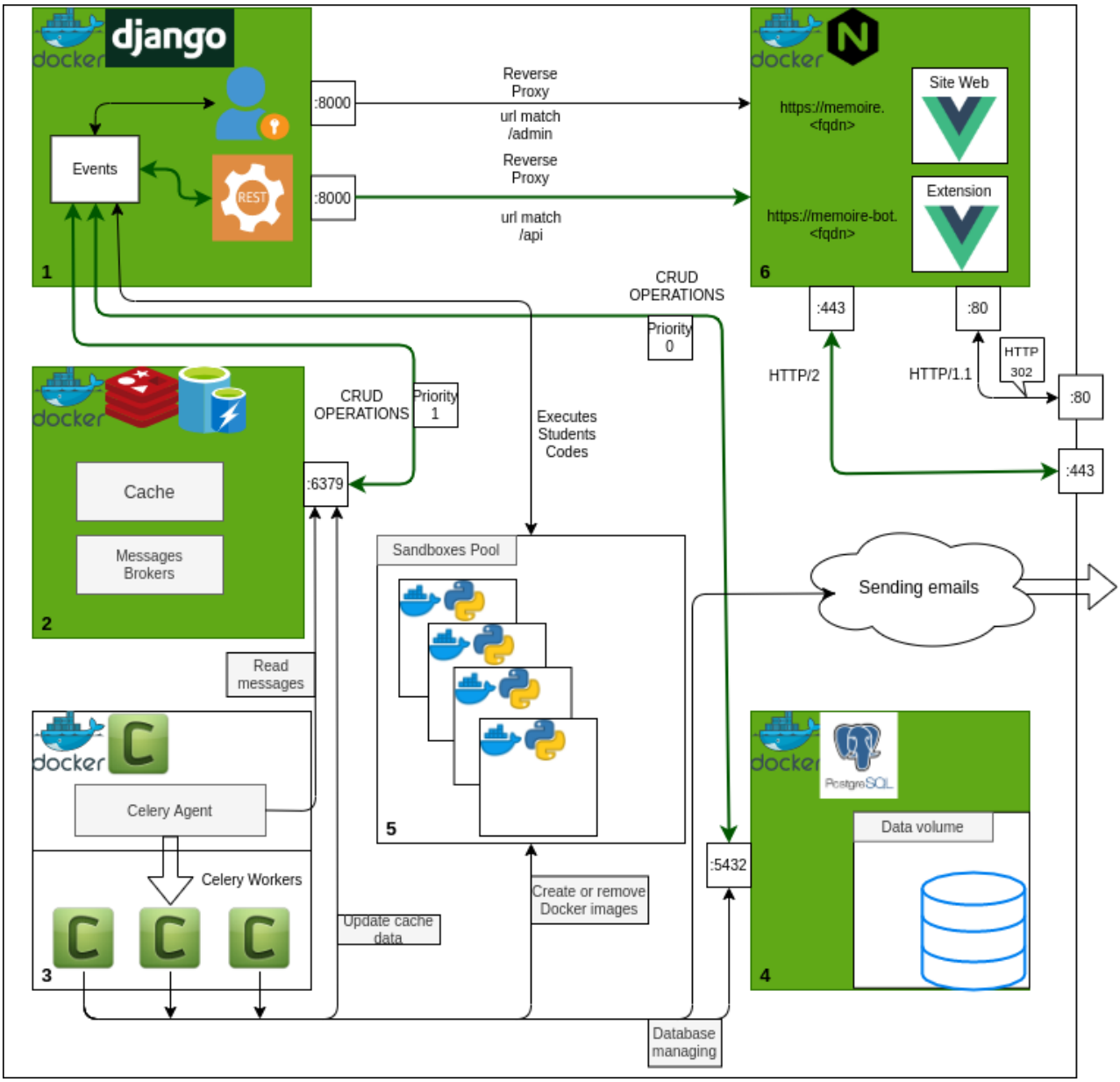


FIGURE 29 – Parties de notre architecture utilisées lors d'une requête de données non-présentes dans le cache.

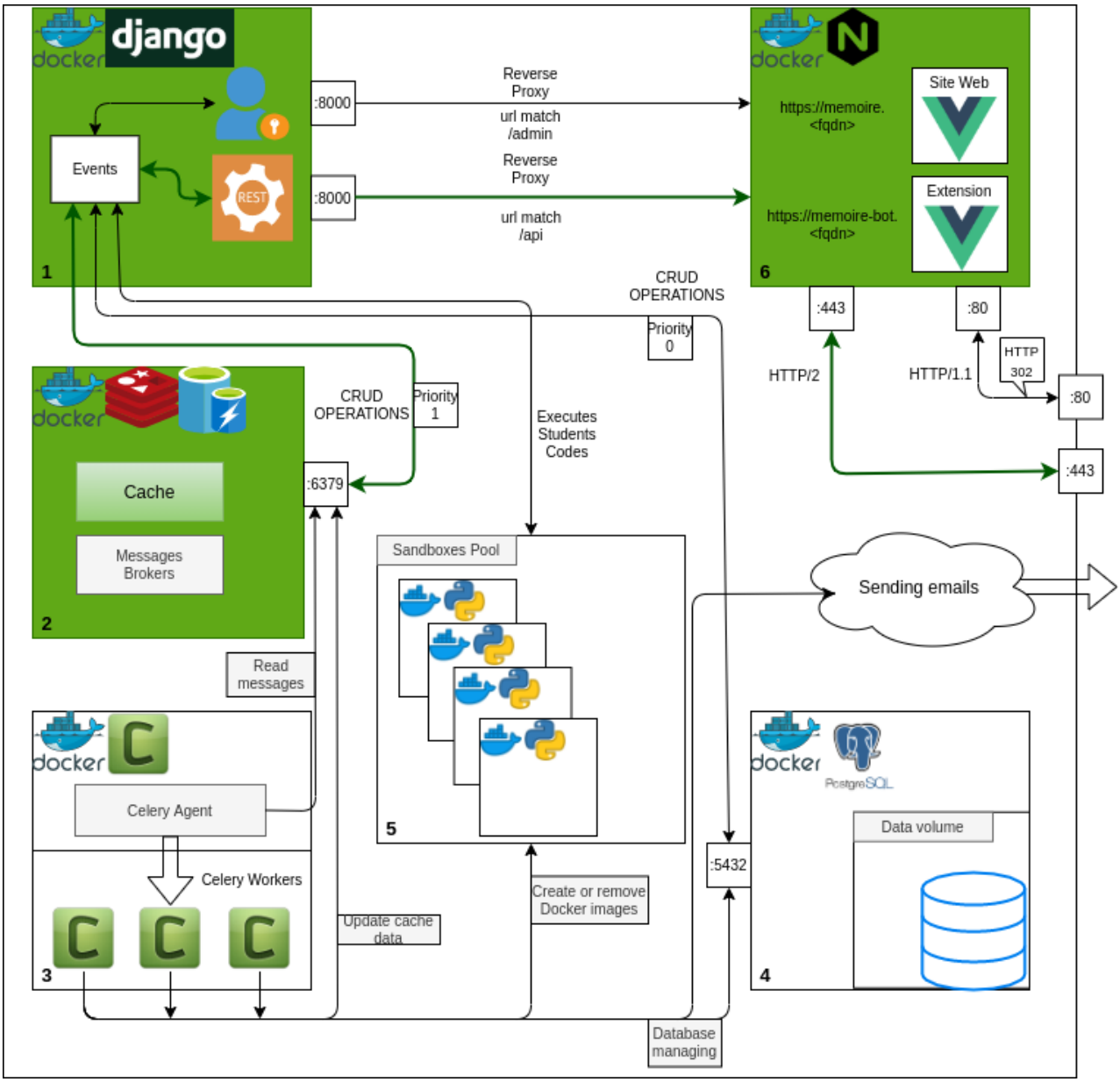


FIGURE 30 – Parties de notre architecture utilisées lors d'une requête de données présentes dans le cache.

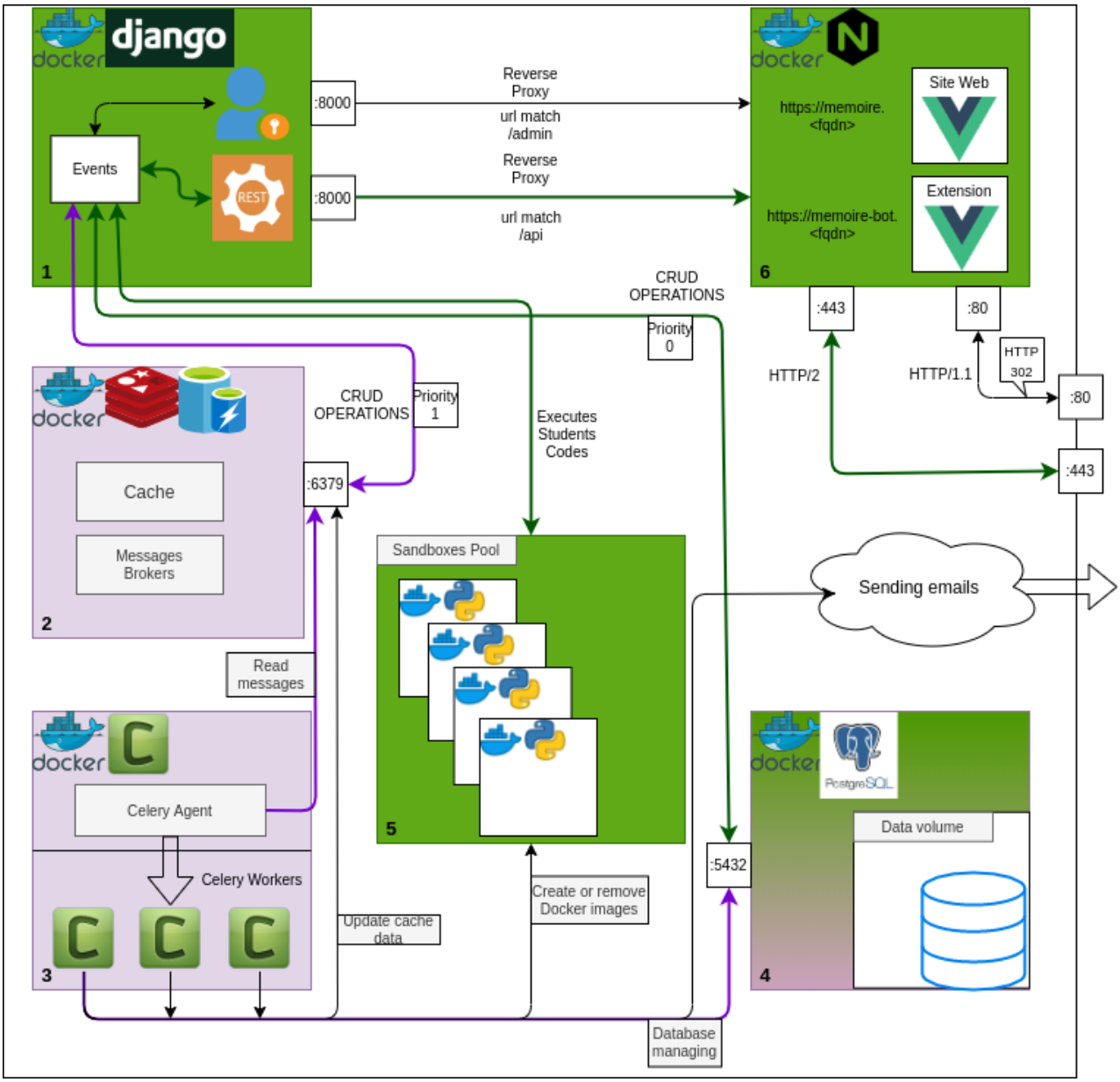


FIGURE 31 – Parties de notre architecture utilisées lors de l'exécution d'un code d'un étudiant.

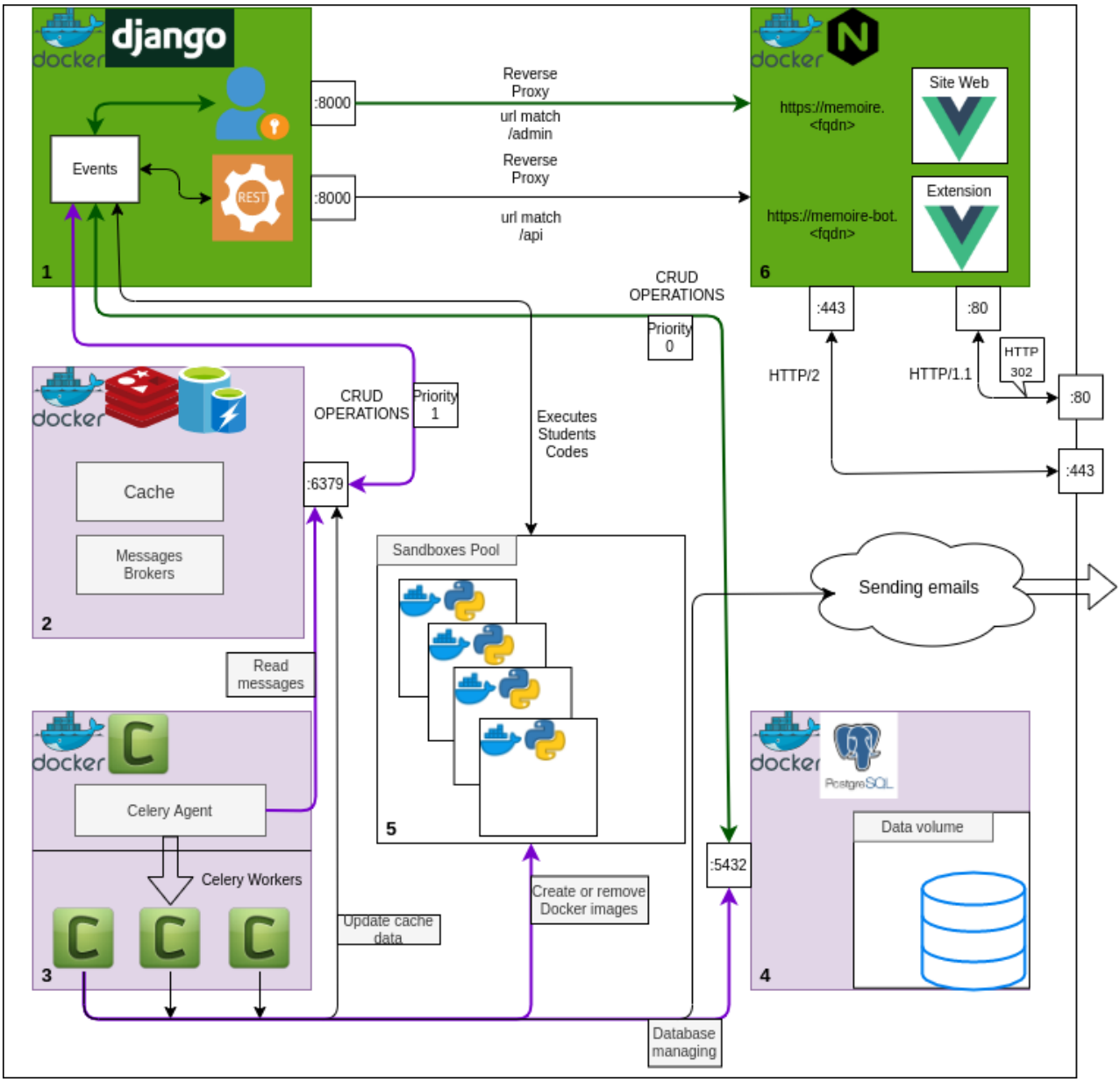


FIGURE 32 – Parties de notre architecture utilisées lors de la création d'un exercice.

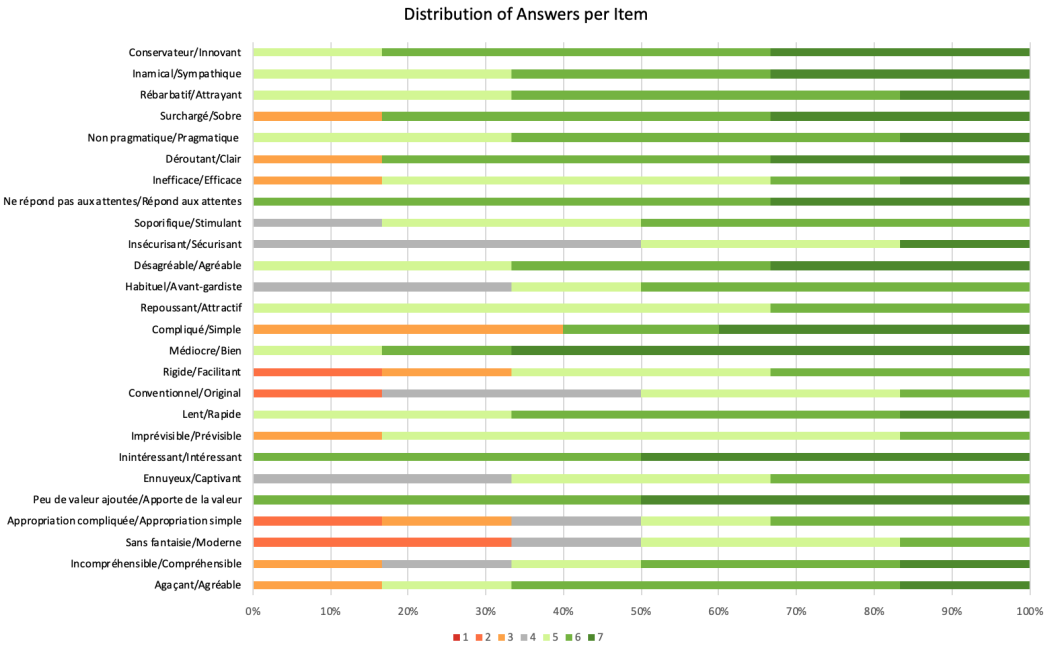


FIGURE 33 – Distribution par question par rapport aux résultats de la section 4.2.

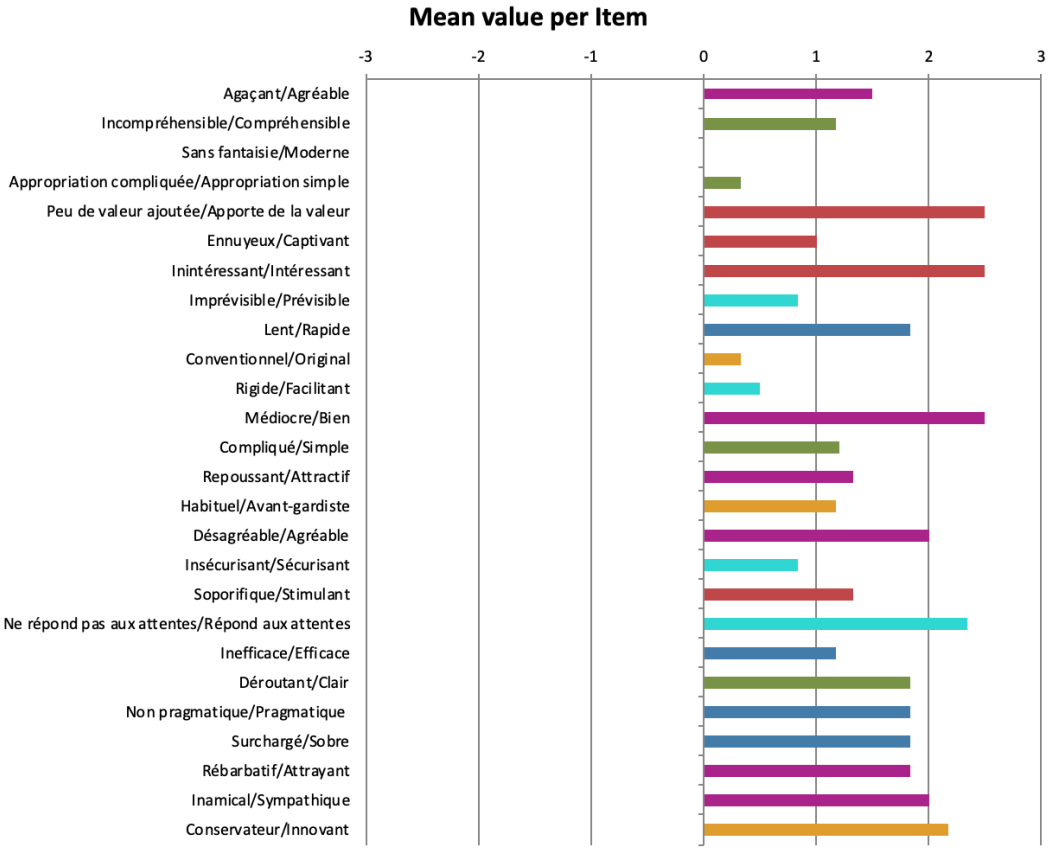


FIGURE 34 – Distribution de la moyenne par question par rapport aux résultats de la section 4.2.

Acronymes

A | B | C | E | G | H | I | J | N | O | P | R | S | T | U | V

A

API Application Programming Interface. 45, 46, 49, 53, 68

ASGI Asynchronous Server Gateway Interface. 45

B

BD Base de Données. 33, 46, 53

C

CRUD Create Read Update Delete. 44

CSS Cascading Style Sheets. 22

E

EDI Environnements de Développement Intégré. 9–11, 20, 30

G

GUI Graphical User Interface. 21

H

HMAC Hash-based Message Authentication Code. 44

HTML HyperText Markup Language. 22

HTTP HyperText Transport Protocol. 45, 47

I

IDE Integrated Development Environment. 3, 12, 20–22, 25, 31, 57

J

JSON JavaScript Object Notation. 45, 46

JWT JSON Web Token. 44

N

NLTK Natural Language ToolKit. 33

NOSQL Not only Structured Query Language. 45, 46

O

ORM Object Relational Mapping. 44, 49

P

POO Programmation Orientée Objet. 21, 81, 84

R

RAM Random Access Memory. 45, 49

REST REpresentational State Transfer. 44, 45

RFC Requests For Comments. 44

S

SGBD Système de Gestion de Bases de Données. 46

SGBDR Système de Gestion de Bases de Données Relationnelles. 44

SGBDRO Système de Gestion de Bases de Données Relationnelles-Objets. 46

SQL Structured Query Language. 44–46, 49, 50

SSL Secure Sockets Layer. 47

T

TP Travaux Pratiques. 25, 32, 40, 50, 66

U

UEQ User Experience Questionnaire. 59

UML Unified Modeling Language. 22

V

VSCode Visual Studio Code. 31, 38, 39, 47, 56

Glossaire

A | B | C | D | F | G | H | I | L | M | N | O | P | R | S | T | V

A

alphabet L'alphabet des langages de programmation comporte de manière classique les lettres de A à Z sans accent, des chiffres et des symboles. 83, 85

architecture En informatique, une architecture correspond à la structure générale d'un système informatique. Elle reprend l'organisation des différents éléments la constituant et les relations entre eux. Une architecture traduit toujours des décisions stratégiques issues d'une phase de conception. 42, 44, 45, 48, 49, 66

B

back-end En informatique, le back-end correspond à tout ce qui est en lien direct avec la couche d'accès aux données ou à l'infrastructure physique. Dans la solution présentée (modèle client-serveur), le back-end correspond au serveur. 38, 41, 44, 55

base de données Une base de données est une structure informatique qui permet de stocker et de retrouver des données brutes ou des informations en rapport à une activité spécifique. 6, 33–35, 39, 45, 46, 48–50, 65

boucle Une boucle est une structure permettant d'exécuter une portion de code plusieurs fois. Ce nombre d'exécutions peut soit être connu à l'avance, soit tant qu'une condition est remplie, le code s'exécutera à nouveau. 8, 18, 19, 27, 29, 35, 48

C

cache Le cache, ou la mémoire cache, est un espace mémoire (physique ou logiciel) qui enregistre des copies de données afin de diminuer le temps d'un accès ultérieur à celles-ci ou d'éviter de devoir les recalculer. 6, 45, 46, 49–51

chaîne de caractères En informatique, une chaîne de caractères est un type de donnée qui consiste en une suite ordonnée de caractères. 6, 33, 35

charge cognitive La charge cognitive est une théorie en psychologie qui cherche à évaluer la capacité de stockage d'informations en mémoire (à court terme), mais également l'intégration de nouveaux éléments. 11–13, 21, 37, 38, 56, 66

chatbot Un chatbot, ou assistant virtuel en français, est un agent qui dialogue avec un utilisateur. 3, 6, 8, 9, 24, 25, 27–39, 41, 55–57, 59, 63, 64, 66–69

classe En Programmation Orientée Objet (POO), une classe regroupe les membres, les fonctions (appelées alors méthodes) et les attributs communs à un ensemble d'objets. 44, 45

commentaire Les commentaires sont des textes qui ne seront pas traduits. Ils sont ajoutés à des programmes afin de laisser des explications. Les commentaires sont délimités par des marqueurs définis par le langage de programmation. 55

compilateur Un compilateur est un programme qui transforme un code source écrit dans un langage de programmation ayant un haut niveau d'abstraction vers un autre langage de plus bas niveau, qui lui est directement exécutable par une machine. 15, 83

condition Une condition en informatique est un test qui va chercher à évaluer une expression logique. Cette condition permet de réaliser des branchements dans un programme. Si le test est vrai, le programme exécutera certaines opérations, sinon le programme exécutera d'autres opérations. 8, 18, 19, 28, 81

console Une console est un périphérique informatique de télécommunications des entrées-sorties d'un système de traitement de l'information. . 3, 32

conteneur Un conteneur ou container, dans un contexte de Docker, est une instance d'une image qui contient, en son sein, tous les processus en cours. 42, 45, 48, 53, 66, 85

D

débogueur Un débogueur ou débugueur est un logiciel qui aide un développeur à analyser les bugs d'un programme. Pour cela, il permet d'exécuter entre autres le programme pas-à-pas. 22

dictionnaire Un dictionnaire est une structure de données qui permet d'associer un ensemble de clés à un ensemble de valeurs. Chaque clé est associée à au plus une valeur (application mathématique). 28, 29

F

fonction En informatique, une fonction est une portion de code réalisant un traitement spécifique et qui renvoie le résultat produit par ce traitement. 81, 84

framework Un framework est un ensemble cohérent de composants logiciels structurels qui sert à créer les fondations ainsi que les grandes lignes d'un logiciel. Un framework a donc pour but d'aider les développeurs en leur facilitant le travail et en augmentant ainsi leur productivité . 41, 42, 44

front-end En informatique, le front-end correspond à tout ce qui est en lien direct avec l'interface et les actions qu'un utilisateur pourra générer. 41, 55

G

gamification Le fait de gamifier. 8, 66, 67, 69

gamifier Action de rendre ludique quelque chose qui ne l'est pas par nature. 66, 82

grammaire formelle Une grammaire est un formalisme permettant de définir des règles de syntaxe et donc un langage. 83, 84

H

HelpDesk Un HelpDesk, ou centre d'assistance en français, correspond au service chargé de répondre aux demandes d'assistance émanant des utilisateurs de produits ou de services. 26–30, 59, 63

I

image Une image, dans un contexte de *Docker*, correspond à un ensemble compilé de structures de fichiers sans les processus. Une image *Docker* contient donc tous les fichiers et programmes que l'on souhaite utiliser. 45–47, 50, 68, 82

interpréteur Un interpréteur est un outil dont la tâche est d'analyser, de traduire et d'exécuter un programme écrit dans un langage de programmation. À chaque exécution du programme, les opérations d'analyse et de traduction seront réalisées, à contrario d'un compilateur. 15, 35, 36, 67, 68

L

langage Un langage formel représente l'ensemble des mots admissibles sur un alphabet donné. 9, 21, 22, 30, 31, 42, 63, 82

langage de programmation Un langage de programmation est créé à partir d'une grammaire formelle, à laquelle on associe des règles de sémantique. Un langage de programmation offre un cadre pour élaborer des algorithmes. 6, 9–11, 21, 22, 25, 27, 30–32, 38, 39, 57, 70, 82, 83

liste Une liste est une structure de données permettant de regrouper des données et de pouvoir accéder à celles-ci de manière libre grâce à un index. 18, 19, 35

M

module Partie du code source d'un logiciel qui peut être développée de manière indépendante. 42, 48, 49, 54

N

notebook Un notebook est un document qui peut contenir du code, des équations, des graphiques ainsi que du texte. Cet outil [53] est fortement utilisé dans le monde scientifique et académique, car il permet de montrer les différentes étapes d'un raisonnement, ce qu'un code seul ne peut pas exprimer. 7, 31, 56

O

objet En informatique, un objet est un conteneur symbolique qui contient des données et des mécanismes, manipulés dans un programme, et qui concerne bien souvent un sujet tangible du monde réel. 44–46, 50, 68, 81, 84

53. https://jupyter.org/

P

pattern Un pattern est une expression régulière destinée à chercher un type de texte donné en respectant certaines contraintes. 24

port En informatique, un port est un élément logiciel qui permet le passage d'informations. 46, 48

procédure En informatique, une procédure est une fonction dont la spécificité est de ne pas renvoyer de valeur. 18

programmation orientée objet Dans le langage informatique, la Programmation Orientée Objet (POO) consiste en la mise en relation d'objets, c'est-à-dire d'éléments de programmation, avec un langage spécifique qui permet aux objets de communiquer entre eux. 22

programme Un programme informatique est un ensemble d'opérations destinées à être exécutées par un ordinateur. Ces programmes peuvent être issus de codes et correspondent à un agencement défini par les règles de syntaxe et par la sémantique. 6, 12, 15, 22, 48, 82–84

R

référence En informatique, une référence est une valeur qui permet l'accès en lecture et/ou en écriture à une donnée. 19, 50

règles de syntaxe Définies par une grammaire formelle, elles régissent les différentes manières dont les éléments du langage peuvent être combinés pour obtenir des programmes. 11, 22, 82, 84

ressource En informatique, les ressources sont des composantes logicielles ou matérielles (fichiers, connexions au réseau ou les espaces en mémoire) connectées à un ordinateur. 32, 48, 50, 67

reverse proxy Un proxy inverse, ou reverse proxy en anglais, est un type de serveur qui permet à un utilisateur d'accéder à des serveurs internes. 45

S

scope Le scope (ou la portée en français) est la région d'un programme informatique où un lien est valide, où le nom peut être utilisé pour faire référence à l'entité. Un lien peut-être une variable, une fonction, etc.. 19, 22

sémantique La sémantique correspond au sens donnés à chacune des phrases qui peuvent être construites dans le langage et plus particulièrement les effets de la phrase lors de l'exécution du programme. 83, 84

serveur Matériel, logiciel ou système informatique destiné à fournir un service déterminé à d'autres systèmes informatiques ou à des utilisateurs connectés sur un réseau. 9, 32, 42, 44–48, 55, 84

structure de données Une structure de données correspond à une manière d'organiser des données afin de pouvoir les manipuler plus facilement. 28, 45, 82, 83

T

threshold le niveau ou le point à partir duquel vous commencez à vivre quelque chose, ou à partir duquel quelque chose commence à se produire ou à changer. 35

V

variable Une variable est une succession de symboles (issues de l'alphabet du langage) permettant d'associer un identifiant à sa valeur. 19, 65, 84

volume Un volume, dans un contexte de Docker, correspond à un dossier partagé entre la machine hôte et le conteneur. On peut également considérer cela comme un point de montage. 47