



> plan-du-rapport

> Introduction

Introduction

Nous arrivons à la fin de notre parcours en MIAGE, après trois années enrichissantes, intenses et pleines d'enseignements, tant professionnels que personnels. Pour clôturer cette année en beauté, nous avons réalisé un projet basé sur une architecture de microservices.

Ce projet nous a permis de développer, en trois mois, une application complète, partant de la réflexion initiale jusqu'aux tests unitaires et à la création du frontend. Ce document en est la preuve et témoigne de la réflexion approfondie sur tout ce que nous avons mis en place, à notre échelle, pour mener à bien ce projet.

Comment mettre en place efficacement un projet de microservices, depuis l'initiation du projet jusqu'à son intégration, tout en garantissant la qualité du code et l'apprentissage des bonnes pratiques du travail en équipe ?

Dans un premier temps, nous détaillerons l'initiation du projet, l'organisation de l'équipe et les choix technologiques qui ont guidé notre travail. Dans la deuxième partie, nous nous concentrerons sur la mise en place concrète de la solution, en expliquant les différentes étapes du développement. Enfin, dans la troisième partie, nous aborderons l'intégration du projet ainsi que les perspectives d'amélioration et les retours d'expérience.

Bien que nous ayons cherché à suivre tous les éléments du projet à la lettre, certains aspects n'ont pas pu être entièrement finalisés en raison de contraintes de temps. Cependant, notre objectif principal a été de produire un code qualitatif et d'apprendre à travailler sur un projet scolaire comme si c'était un projet en entreprise.

Edit this page



1- Initialisation du projet

a) Genèse et objectifs : Contexte et enjeux du projet

L'objectif principal de ce projet était de le traiter comme un véritable projet d'entreprise. Il s'inscrit dans le cadre du cours d'interopérabilité, mais mobilise en réalité l'ensemble des enseignements acquis tout au long du parcours en MIAGE. Nous avons cherché à appliquer des concepts théoriques à un cas pratique concret, en respectant les contraintes techniques et les exigences imposées par le cahier des charges.

Afin de rendre l'expérience plus ludique et motivante, nous avons choisi de développer une plateforme d'escape game en ligne, sur le thème de la MIAGE. Ce choix, à la fois original et en lien avec notre cursus, a permis de garder une dimension fun et engageante, même lors des moments de challenge technique. L'idée était de créer un projet qui allie l'utile à l'agréable, et qui soit suffisamment motivant pour l'ensemble de l'équipe tout au long du développement.

Notre vision du projet était de créer une plateforme innovante qui reflète l'esprit de la MIAGE, tout en offrant une expérience immersive et divertissante. L'escape game en ligne était une manière de rendre la mise en pratique des concepts abordés pendant la formation plus interactive et engageante, tout en intégrant des mécanismes de sécurité et de gestion de données réels. Nous avons souhaité construire un projet qui pourrait être utilisé non seulement comme une application fun, mais aussi comme une véritable démonstration de notre expertise technique et de notre capacité à gérer des projets complexes.

- Les consignes imposées étaient les suivantes :
 - Une architecture microservices pour une meilleure modularité et évolutivité.
 - La mise en place de mécanismes de sécurité et d'authentification, afin de garantir la protection des données utilisateurs et des interactions sur la plateforme.
 - L'implémentation d'un service en .NET, afin d'appliquer et d'expérimenter une technologie différente de celle que nous utilisons habituellement (Spring Boot).

- La nécessité de tester l'application, pour garantir la qualité du code et la fiabilité des fonctionnalités mises en place.
- L'utilisation obligatoire de Docker, pour assurer la portabilité et la reproductibilité de l'environnement de développement.

Dans le but de construire la base de notre escape game et de poser les fondations du cœur de notre projet, nous avons fait le choix d'architecturer notre système autour de microservices indépendants. Ces microservices interagissent de manière indirecte, favorisant ainsi la modularité et la scalabilité de notre application. Parmi ces services.

- On retrouve notamment :

- Le microservice authentification : Ce microservice joue un rôle central puisqu'il est chargé de la gestion des comptes utilisateurs et de leurs données personnelles. Il permet à un nouvel utilisateur de s'inscrire sur la plateforme, puis de se connecter à son compte. Une fois connecté, l'utilisateur peut consulter et modifier ses informations personnelles. Ce microservice prend également en charge la réinitialisation du mot de passe, grâce à un système de vérification et de validation du code envoyé à l'utilisateur. Enfin, il offre la possibilité de créer un compte administrateur, disposant de droits pour superviser et gérer l'ensemble de l'application.
- Le microservice Jeu : Ce microservice est dédié à la gestion de l'escape game dans sa globalité. Il se compose de trois contrôleurs, chacun responsable d'un aspect essentiel du jeu : la gestion des parties, des énigmes, et de la question finale (ou "question master"). Le contrôleur en charge des parties permet notamment de démarrer et de terminer une session de jeu, assurant ainsi le bon déroulement de l'expérience utilisateur. Du côté des énigmes, le microservice offre une gestion complète allant de leur création jusqu'à leur suppression. Enfin, il fournit également des fonctionnalités liées à la consultation des indices disponibles ainsi qu'au suivi du nombre de tentatives effectuées pour répondre à la question finale, concluant ainsi la partie. L'utilisateur a également la possibilité de faire remonter ses problèmes ou ses questions grâce à un système de gestion des retours, qu'il s'agisse de feedbacks ou de suggestions.
- Le microservice actualité : Ce microservice est chargé de la gestion des publications sur la plateforme, communément appelées posts. Il offre un ensemble de

fonctionnalités permettant de créer, consulter, modifier ou supprimer des posts. En complément de cette gestion, le microservice permet également aux utilisateurs d'interagir avec les publications en y réagissant à travers différentes réactions disponibles.

- Le microservice récompense : Ce microservice est dédié à la gestion des éléments liés à la valorisation et au classement des utilisateurs sur la plateforme. Il regroupe principalement deux aspects fonctionnels : la gestion des badges et celle du classement. À travers ses différents contrôleurs, le service permet la gestion des badges attribués aux utilisateurs. Ces badges, symboles de progression ou de réussite, sont attribués automatiquement par RabbitMQ. Par ailleurs, ce microservice expose également des endpoints permettant de consulter les classements globaux ou hebdomadaires, offrant ainsi une visibilité sur la performance des joueurs.
- Le microservice notification : Ce microservice est chargé de la gestion de l'ensemble des envois de courriels au sein de la plateforme. Son rôle est d'assurer la communication par mail avec les utilisateurs à différents moments clés et dans divers cas d'usage. Il permet notamment l'envoi d'un mail de bienvenue lors de l'inscription, d'un mail d'anniversaire le jour concerné, ou encore d'un mail de confirmation suite à une réponse reçue à un feedback précédemment envoyé. Ce service prend également en charge l'envoi de mails de réinitialisation de mot de passe ainsi que l'envoi régulier de statistiques hebdomadaires.

Ce projet a donc permis de conjuguer plusieurs compétences techniques tout en suivant un processus de développement Agile, avec une forte collaboration entre les membres de l'équipe. En choisissant de créer une plateforme d'escape game, nous avons réussi à allier travail sérieux et motivation personnelle, ce qui nous a permis de maintenir notre enthousiasme et notre investissement, même dans les moments les plus complexes du développement.

b) Organisation et outils de gestion

Étant donné le temps limité et les défis rencontrés, une organisation rigoureuse était essentielle. Nous avons structuré l'équipe en deux sous-groupes : backend et frontend. Cette division nous a permis de concentrer nos efforts tout en assurant une communication

constante via Discord, surtout lors des points réguliers pour ajuster notre travail et avancer rapidement.

a- Mise en place de l'organisation

Lors des premières réunions, nous avons rapidement défini notre vision du projet et clarifié nos objectifs. Cela a débouché sur la mise en place d'un tableau d'URLs et d'un schéma d'architecture que nous détaillerons plus tard. Pour organiser nos tâches sur le long terme, nous avons opté pour Trello, ce qui a facilité le suivi de l'avancement et assuré une répartition claire du travail.

b- Gestion du code avec GitLab

Initialement sur Pdicost, nous avons migré notre code vers GitLab pour bénéficier d'une meilleure gestion avec CI/CD et Merge Requests. Cette migration nous a permis de structurer le travail avec des branches dédiées et des revues de code régulières, garantissant ainsi une gestion rigoureuse du code.

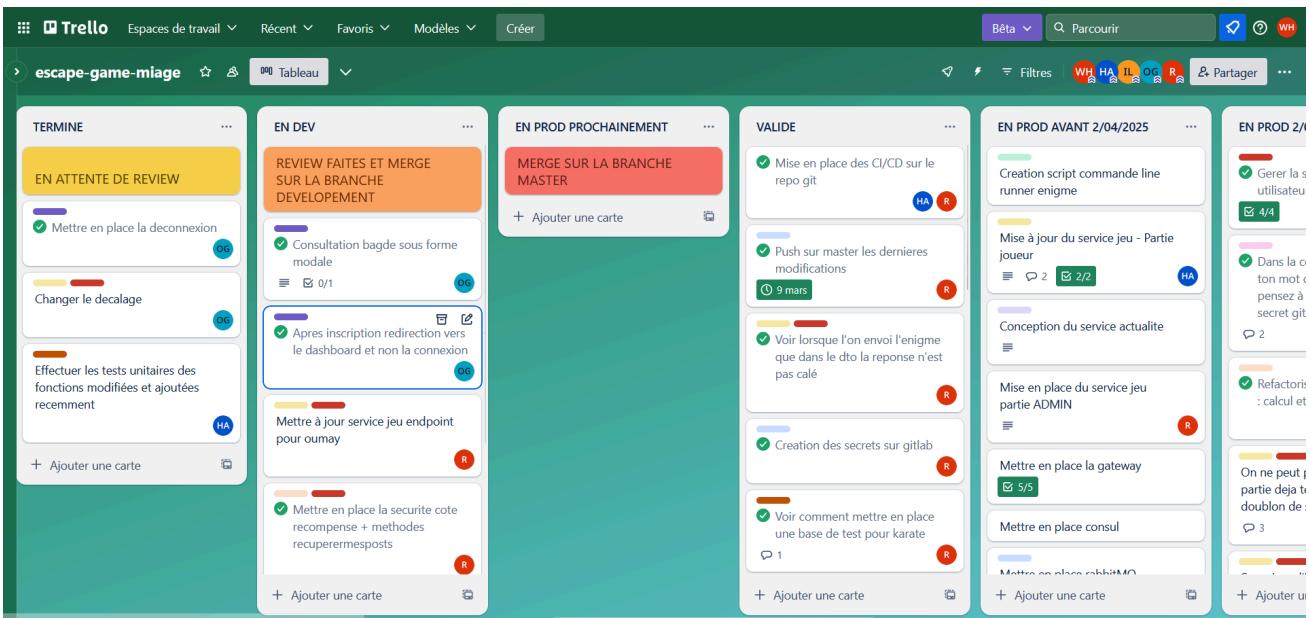
Les merge requests ont véritablement été un atout majeur tout au long du développement du projet. Elles ont permis d'instaurer un processus de vérification rigoureux avant chaque intégration de code, garantissant ainsi la stabilité et la cohérence de l'ensemble du système. Grâce à elles, chaque membre de l'équipe pouvait proposer des modifications du code sur sa branche tout en bénéficiant de retours constructifs de la part des autres développeurs. Ce système de revue de code a non seulement permis de corriger les éventuelles erreurs ou incohérences en amont, mais aussi d'échanger sur les bonnes pratiques et d'améliorer continuellement la qualité du code. En somme, les merge requests ont été un outil essentiel pour maintenir une dynamique collaborative et viser un code propre, lisible et maintenable.

c- Qualité du code

L'objectif principal côté backend était de produire un code propre et maintenable, en respectant les bonnes pratiques de développement vues en MIAGE. Nous avons intégré des tests unitaires et veillé à la documentation adéquate, pour faciliter l'intégration avec le frontend et maintenir une qualité de code constante tout au long du développement.

Lien vers le Trello : [escamiage-Trello](#)

Une capture du Trello pendant la dernière semaine du projet :



Un exemple d'une Issue créée sur GitLab :

[BACK] - maj service actu

[Open](#) [Issue created 2 days ago by Romaissa](#)

- corriger methode recuperer mes posts
- endpoint getpostbyid

[0](#) [0](#) [0](#)

[Add design](#) [Create merge request](#)

Child items 0

No child items are currently assigned. Use child items to break down work into smaller parts.

Linked items 0

Link items together to show that they're related or that one is blocking others.

Development

[Resolve "\[BACK\] - maj service actu"](#) 157 [Merged](#) [X](#) [New](#) [More](#)

Activity

Romaissa added **back**, **bugfix** labels 2 days ago

Romaissa assigned to [@rberrekam](#) 2 days ago

[All activity](#) [Oldest first](#)

Assignee Romaissa [Edit](#)

Labels **back** **bugfix** [Edit](#)

Dates Start: None Due: None [Edit](#)

Milestone None [Edit](#)

Parent None [Edit](#)

Time tracking Add an estimate or time spent. [Edit](#)

Contacts None [Edit](#)

Participant [Edit](#)

c) Choix des technologies

Le cahier des charges spécifie que l'architecture devait reposer sur cinq microservices. Quatre d'entre eux sont développés en Spring Boot et utilisent Spring Security ainsi que JPA pour la gestion des données. Un des services est quant à lui développé en .NET et plus précisément sous forme de worker. Le choix du framework frontend s'est porté sur Angular,

en raison de son efficacité pour le développement d'applications web dynamiques et réactives.

En ce qui concerne les bases de données, nous avons opté pour MySQL en raison de ses performances, de sa robustesse et de sa compatibilité avec notre architecture. Afin de garantir la qualité et la fiabilité du système, des tests unitaires ont été mis en place à l'aide de JUnit et Mockito. De plus, des tests d'intégration ont été réalisés en utilisant le framework Karate, permettant ainsi de tester l'ensemble des interactions entre les composants.

Tous les services, y compris le frontend, ont été dockerisés. Ils sont lancés via Docker Compose, qui orchestre également les bases de données, permettant une gestion simplifiée et cohérente de l'environnement. Concernant la communication entre les microservices, RabbitMQ a été intégré comme système de messagerie. De plus, un autre service Spring a été mis en place en tant que load balancer / gateway. Ce service centralise les demandes des utilisateurs, les redirigeant vers les microservices appropriés, et assure ainsi la gestion des requêtes et la haute disponibilité du système.

Enfin, pour gérer la découverte des services, chaque microservice, au démarrage, s'enregistre sur Consul, permettant ainsi une gestion dynamique de l'infrastructure. Nous détaillerons plus en profondeur les interactions entre ces composants dans les sections suivantes, cette partie servant simplement d'introduction aux choix technologiques.

 [Edit this page](#)



2- Développement et mise en oeuvre

a) Architecture et conception

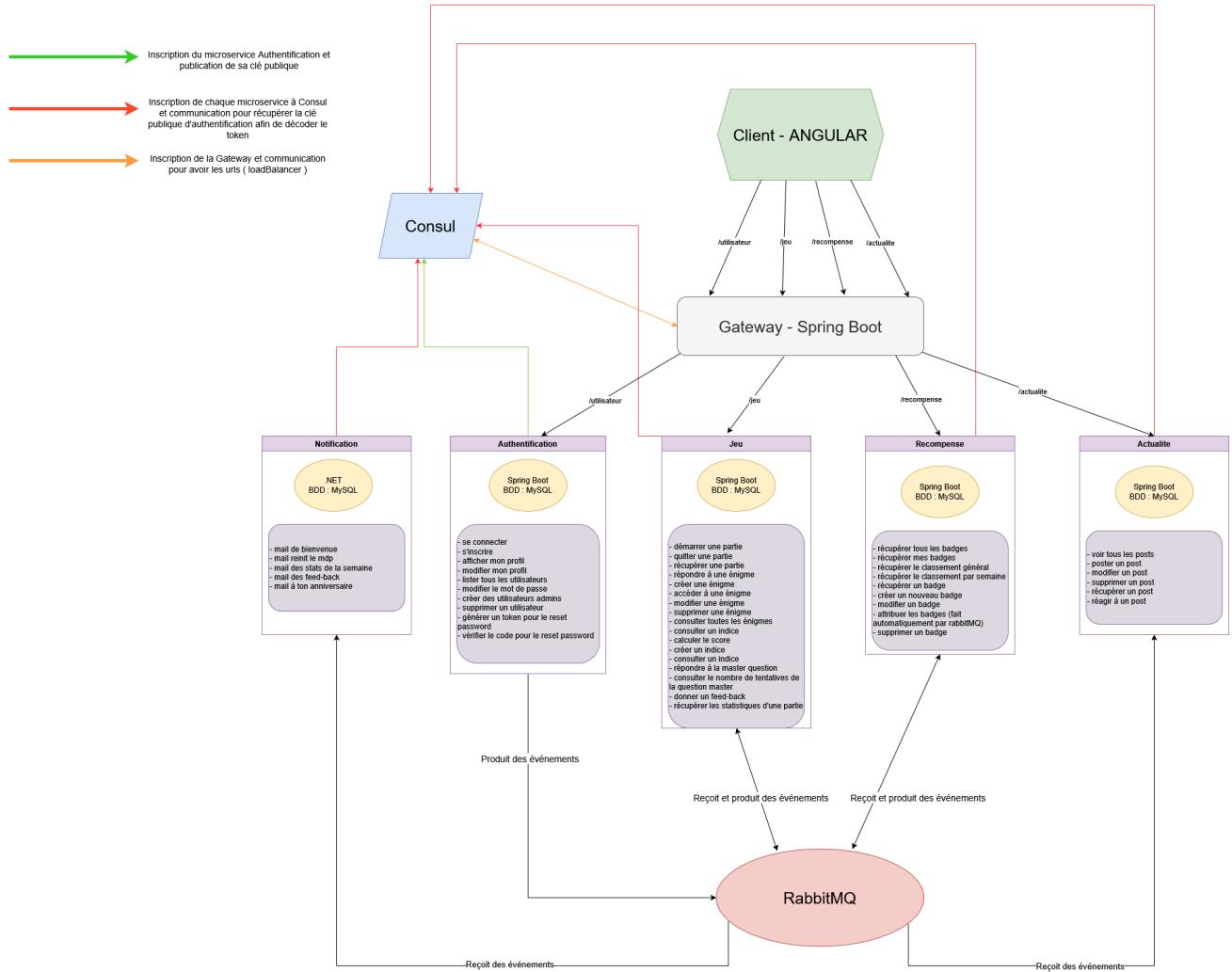
a. Découpage des services et interactions

Le découpage des services a été une première étape complexe, car il ne fallait pas trop découper les fonctionnalités tout en respectant les principes d'une architecture microservices. Étant donné que le projet porte sur un escape game en ligne, nous avons dû réfléchir à la manière d'organiser les différents services tout en gardant à l'esprit la modularité et la scalabilité de l'application. Nous avons donc identifié plusieurs services distincts : une partie gestion par l'administrateur, une partie inscription et connexion des utilisateurs, une partie dédiée au jeu lui-même où les utilisateurs peuvent interagir et jouer, et enfin une partie pour la gestion des données administratives et des sessions de jeu.

b. Modèle de données

L'architecture du projet est une étape cruciale dans la mise en place du système, surtout avec l'intégration de 5 microservices qui communiquent entre eux. Il était donc essentiel de définir un modèle de données clair et cohérent. Nous avons choisi d'intégrer ce modèle directement dans le schéma d'architecture pour visualiser plus facilement les interactions entre les différents services.

Le schéma des interactions avec et entre les différents services :



Le modèle des données :

Entité(attributs ou/et clés primaires/étrangères)	Microservice lié
utilisateur(id, created_at, date_naissance, email, image, nom, password, prenom, pseudo, role, updated_at)	Authentification ▾ clé primaire - #clé étrangère
utilisateur_seq(next_val_)	Authentification ▾
enigme(id, difficulté, indice, question, réponse, thème)	Jeu
partie(id, code, date_creation, état, indice_code, nb_retentative_code, score_final, temps_alloué, temps_final, #utilisateur_id)	Jeu
réponse(id, nombre_tentative, répondre, score, #enigme_id, #partie_id)	Jeu
utilisateur(id, email, pseudo)	Jeu
badge(id, condition_type, condition_value, icone, nom)	Actualité ▾
badge_utilisateur(id, #utilisateur, #badge_id)	Actualité ▾
post(id, #auteur, contenu, date_publication, image, type)	Actualité ▾
post_seq(next_val_)	Actualité ▾
reaction_utilisateur(id, reaction_post, #post_id, #utilisateur_id)	Actualité ▾
reaction_utilisateur_seq(next_val_)	Actualité ▾
template_post(id, contenu_template, type_badge)	Actualité ▾
utilisateur(id, email, pseudo, score_semaine)	Actualité ▾
utilisateur_mes_badges(#utilisateur_id, #mes_badges_id)	Actualité ▾
badge(id, condition_type, condition_value, date_creation, date_modification, icone, nom)	Récompense ▾
badge_seq(next_val_)	Récompense ▾
badge_utilisateur(#utilisateur_id, #badge_id)	Récompense ▾
partie(id, date_partie, durée_partie_en_secondes, #id_utilisateur, is_reussie, nb_tentatives_total, réponses_correctes, score)	Récompense ▾
partie_themes(#partie_id, theme)	Récompense ▾
statistiques(id, #id_utilisateur, durée_moyenne_partie, meilleur_temps, nb_parties_jouées, nb_tentatives_global, pire_temps, score_semaine)	Récompense ▾
statistiques_seq(next_val_)	Récompense ▾
statistiques_themes(#statistiques_id, theme)	Récompense ▾
utilisateur(id, date_dernière_partie, meilleur_score, meilleur_temps, nb_parties_jouées, score_semaine)	Récompense ▾

Un aperçu du tableau des URIs :

Microservice associé	/escamimage	request webservice			response			
		fonctionnalités	URI	Méthode	Contenu	Status	Cause (raison fonctionnelle)	Contenu éventuel
Authentification	S'inscrire	/utilisateur	POST	BODY UtilisateurDTO		201	Utilisateur créé avec succès	token url vers la ressource créée (Location)
						400	Requête invalide - Les données fournies sont incorrectes ou mal formatées	400
						409	Conflit - Un utilisateur avec cet email existe déjà	409
Authentification	S'authentifier	/utilisateur/connexion	POST	BODY email, password		200	Authentification réussie	token
						400	Requête invalide - Les données fournies sont incorrectes ou mal formatées	400
						401	Non autorisé	401
Authentification ADMIN	Voir tous les utilisateurs	/utilisateur	GET	HEADER token		200	Succès	[[UtilisateurDTO], ...]
						401	Non autorisé - L'utilisateur n'est pas authentifié	401
						403	Interdit - L'utilisateur n'a pas les droits d'administrateur	403
Jeu ADMIN	Créer une énigme	/jeu/enigmes	POST	HEADER token BODY EnigmeDTO		201	L'énigme a bien été créée	url vers l'énigme (Location)
						400	Requête invalide - Les données fournies sont incorrectes ou mal formatées	400
Jeu ADMIN	Modifier une énigme	/jeu/enigmes/{idEnigme}	PUT	HEADER token BODY EnigmeDTO		200	La mise à jour a bien été faite	EnigmeDTO
						400	Requête invalide - Les données fournies sont incorrectes ou mal formatées	400
						404	L'énigme n'existe pas	404
Jeu ADMIN	Consulter toutes les énigmes	/jeu/enigmes	GET	HEADER token		200	La liste des énigmes a été récupérée avec succès	[[EnigmeDTO], ...]
						401	Un problème d'authentification est survenu	401

Lien vers le tableau des URIs et le modèle de données : [tableau-des-uri-et-modele-de-donnees](#)

Un aperçu des premières maquettes réalisées :

The figure displays three screenshots of the application's user interface:

- Welcome Screen:** Shows a dark background with various items like a clock, a globe, and papers scattered around. A central box contains the text "Bienvenue sur escape game !" and a "COMMENCER" button.
- Inscription Screen:** Shows a similar cluttered background. A central form asks for "Nom", "Prénom", "Email", and "Mot de passe". It also includes a "Confirmer le mot de passe" field and a "S'inscrire" button. Below the form, it says "Créez votre compte en quelques secondes." and "Déjà un compte ? Se connecter".
- Accueil ADMIN Screen:** Shows a sidebar with "ESCAMIAGE", "Mes enigmes" (which is selected), "Ajouter enigmes", and "Afficher utilisateurs". The main area lists "Nom Admin" and "4 enigmes trouvées". Below this, there are four entries for puzzles, each with fields for "Description", "Réponses", and buttons for "Modifier" and "Supprimer". At the bottom right, there is a "Frame 1" button with "Votre profil" and "Déconnexion" options.

b) Implémentation

a. Backend

API REST

Pour le backend, nous avons adopté une architecture API REST permettant à chaque service de gérer des requêtes HTTP de manière claire et structurée.

SPRING BOOT

Pour assurer la sécurité de l'application, Spring Security a été utilisé pour gérer l'authentification et l'autorisation des utilisateurs, et nous avons intégré JWT (JSON Web Tokens) pour sécuriser les échanges entre le client et les différents services.

Communication avec RabbitMQ

Les services sont organisés sous forme de microservices, chacun étant responsable d'une fonctionnalité spécifique. Grâce à l'utilisation de RabbitMQ, les microservices échangent des messages asynchrones pour traiter les tâches et coordonner les actions sans dépendre d'appels synchronisés bloquants. Chaque service est autonome, ce qui permet de maintenir une architecture modulaire, facilement scalable et flexible.

Les microservices backend communiquent entre eux via RabbitMQ, un système de messagerie asynchrone qui facilite l'échange d'informations entre les services de manière fiable et scalable. Cela permet une gestion fluide de la communication inter-service, en déchargeant les services des appels synchronisés bloquants et en garantissant une meilleure performance.

Consul

Worker .NET

Le service de notification est développé en .NET. Étant donné qu'il ne fait qu'envoyer des e-mails — c'est-à-dire des actions s'exécutant en arrière-plan en réponse à des messages asynchrones (par exemple via RabbitMQ), ou pour réaliser des tâches planifiées comme l'envoi d'e-mails d'anniversaire — le Worker Service proposé par .NET est parfaitement adapté à ce type de besoin.

Des exemples mails gérés par ce service : Mail le jour de l'anniversaire :

20:03

37



...

Joyeux Anniversaire !

Inbox



Sougouma Ali... 18 Mar
to me ▾



...



Translate to English



Joyeux Anniversaire Sougouma Ali !

Toute l'équipe d'EscapeGame vous souhaite
un merveilleux anniversaire pour vos 0 ans.

Profitez bien de cette journée spéciale !

Reply

Forward



Mail des statistiques hebdomadaire :

20:00

42



...

Statistiques de la semaine

Inbox



Sougouma Ali... 18 Mar
to me ▾



...



Translate to English



Bonjour Sougouma Ali Hamid,

Voici vos statistiques de la semaine :

- Meilleur temps : 1200 secondes
- Meilleur score : 3500
- Date de la dernière partie : 16/03/2025 00:00:00
- Nombre de parties jouées : 10
- Score de la semaine : 150



Sougouma Ali... 18 Mar
to me ▾



...



Translate to English



...

Mail de réinitialisation du mot de passe :

20:01

42



...

Réinitialisation de votre mot de passe



Inbox



Sougouma Ali... 11 Mar
to me ▾



...



Translate to English



Bonjour Sougouma Ali Hamid,

Vous avez demandé une réinitialisation de
votre mot de passe.

Utilisez le token suivant pour confirmer:

792415

Si vous n'avez pas demandé cette
réinitialisation, veuillez ignorer cet email.

2



Sougouma Ali HAMID 11 Mar
Bonjour Sougouma Ali Hamid, Vous ave...



Sougouma Ali... 11 Mar
to me ▾

b. Frontend

Pour le frontend, nous avons opté pour Angular, un framework puissant et flexible pour créer des applications web dynamiques. Le projet a été structuré en composants Angular permettant une gestion modulaire de l'interface utilisateur. Les données sont récupérées via

des appels HTTP aux API backend et via des messages RabbitMQ pour assurer une communication fluide avec le backend.

La structure des composants de l'application est pensée de manière modulaire et fonctionnelle, favorisant la lisibilité, la maintenabilité et l'évolutivité du projet. Chaque fonctionnalité ou domaine métier est isolé dans un dossier propre situé sous `src/app`, ce qui permet un découplage logique du code et facilite le travail en équipe. Par exemple, le dossier `acces` regroupe tous les composants liés à l'authentification des utilisateurs, notamment `login` et `register`, responsables de la connexion et de l'inscription. Le module `actualite` est plus riche, car il gère toute la logique autour des publications : création, affichage, modification, réaction aux posts, ainsi que les repositories et services permettant d'assurer la persistance et l'interaction avec le backend.

De manière similaire, le dossier `badge` centralise les composants permettant la création, visualisation et modification des badges, avec une séparation claire entre la logique métier et la présentation via les services et repositories dédiés. Le module `enigme` est plus ciblé, avec des composants orientés vers la gestion des énigmes, leur création et leur édition. Le dossier `jeu`, quant à lui, regroupe l'ensemble des composants nécessaires à la logique du jeu, à sa page de règles et à la communication avec les données via services et repositories. Cela permet de maintenir une logique métier cohérente.

Le module `user` contient plusieurs sous-fonctionnalités comme l'affichage des utilisateurs, la gestion des feedbacks et suggestions, le dashboard utilisateur, ainsi que le profil, qui est suffisamment riche pour être extrait dans un sous-module à part entière. Le dossier `profil` encapsule les composants liés à l'affichage et à la modification du profil utilisateur, tout en disposant de ses propres services et repositories pour gérer la logique métier de manière indépendante.

Cette organisation favorise une approche scalable et modulaire. Elle permet également une réutilisabilité des composants, une séparation des responsabilités, et s'aligne avec les bonnes pratiques des architectures modernes orientées composants.

c. Documentation API avec Swagger

La documentation de l'API a été réalisée avec Swagger, qui génère automatiquement une interface interactive et permet de tester directement les différents endpoints de l'API. Cela facilite la compréhension de l'API par d'autres développeurs et permet une intégration plus rapide et plus fluide des services.

c) Tests et assurance qualité

a. Test unitaires (JUnit, Mockito)

Pour garantir la fiabilité du code, nous avons mis en place des tests unitaires à l'aide de JUnit, combinés avec Mockito pour le mocking des dépendances externes. Ces tests ont été réalisés pour chaque méthode et composant du backend afin de vérifier leur bon fonctionnement de manière isolée. L'objectif était de s'assurer que chaque fonction se comporte comme prévu dans un environnement contrôlé, sans interagir avec des composants externes ou des bases de données réelles.

Avec Mockito, nous avons pu simuler des comportements de dépendances (par exemple, des services externes ou des objets complexes) et vérifier que les interactions avec ces dépendances se passent comme prévu. Cela nous a permis de tester la logique interne des composants sans avoir besoin d'implémenter des appels réels, rendant les tests plus rapides et plus ciblés.

b. Tests d'intégration (Karaté)

Pour les tests d'intégration, nous avons utilisé Karaté pour tester l'interaction entre les différents microservices. Ces tests se sont concentrés sur la communication entre les services via la gateway, en passant par les endpoints exposés par cette dernière. Contrairement aux tests unitaires qui isolent les composants, ces tests simulent des appels réels entre les microservices, en vérifiant que l'ensemble du système fonctionne comme prévu.

Les tests d'intégration ont permis de vérifier le bon fonctionnement des flux de données entre les services, tout en simulant des requêtes HTTP réelles et en validant les réponses des microservices. Cela a garanti que les microservices interagissent correctement, en tenant compte des différents cas d'usage dans un environnement de production.

Pour éviter d'impacter la base de données en production, une base de données de test a été mise en place, permettant de simuler des interactions réelles sans toucher aux données réelles du système.

Cette base est une base de données MySQL et est initialisée avec un jeu de données de test. Lors du démarrage des tests, le schéma de la base est généré automatiquement, puis des

données pertinentes sont insérées pour permettre de valider les interactions entre les composants applicatifs et la base de données.

Un docker compose spécifique pour cette effet a été crée afin de lancer les spring boot avec le profil test, pour garantir un fonctionnement indépendant et une gestions plus précise des fichiers de test.

 [Edit this page](#)



3- Intégration et perspectives

a) Mise en place de GitLab CI/CD

Nous avons mis en place une pipeline CI/CD avec GitLab pour automatiser les étapes de construction, de test et de déploiement de notre application. Cette approche nous permet de garantir que chaque modification du code est immédiatement testée, et que les versions déployées sont toujours stables. Grâce à cette intégration continue, nous avons pu détecter rapidement les erreurs et maintenir une qualité constante du code tout au long du développement.

Fonctionnement de la pipeline :

- Tests unitaires : À chaque Merge Request, les tests unitaires sont automatiquement lancés pour vérifier le bon fonctionnement des différentes méthodes et composants du backend.
- Tests d'intégration : De même, les tests d'intégration sont exécutés afin de garantir que la communication entre les services via la gateway fonctionne comme prévu.
- Gestion des secrets : Pour garantir la sécurité des informations sensibles (comme les mots de passe, les clés API, etc.), nous avons mis en place un système de gestion des secrets, par exemple pour le Docker Compose. Bien que nous soyons en environnement local, nous avons anticipé la nécessité de sécuriser ces informations en prévision de futures mises en production.
- Déploiement automatique sur le dépôt principal : Une fois que les tests sont réussis et que les modifications sont validées, le code est automatiquement poussé sur la branche master de notre dépôt, via une pipeline dédiée.

Cette pipeline CI/CD garantit non seulement la qualité du code, mais aussi une gestion sécurisée des informations sensibles et un déploiement fluide des nouvelles versions de

l'application.

b) Lancement du projet avec Docker et le README

Le projet a été configuré pour être facilement exécuté en local, en version de développement, via Docker. Un README détaillé a été fourni pour guider les utilisateurs dans le processus de mise en place de l'environnement local.

Ce guide inclut les instructions nécessaires pour récupérer le projet, configurer Docker et lancer les différents services de l'application (backend, frontend, bases de données, etc.). Ce processus vise à permettre à tout développeur ou utilisateur de tester rapidement le projet dans son environnement local, sans avoir à se soucier de la configuration des services ou de l'infrastructure.

c) Problèmes rencontrés et solutions apportées

L'une des principales difficultés rencontrées durant cette phase a été de définir correctement les variables d'environnement et de s'assurer que tous les services fonctionnaient ensemble via Docker. Nous avons dû faire quelques ajustements dans le fichier docker-compose.yml pour garantir que tous les services se connectent correctement entre eux et avec la base de données. Ces ajustements ont été documentés dans le README pour faciliter l'expérience des utilisateurs.

d) Axes d'améliorations

Parmi les axes d'amélioration identifiés, la mise en place d'un mécanisme de refresh token constitue une évolution importante du système d'authentification. Actuellement, l'accès repose uniquement sur un token d'authentification classique, ce qui limite la gestion fine des sessions utilisateurs. L'intégration d'un refresh token permettrait de prolonger les sessions de manière sécurisée, sans avoir à redemander les identifiants de l'utilisateur, tout en renforçant la sécurité via un cycle de vie contrôlé des tokens.

En parallèle, dans une logique DevOps et de fiabilisation des processus, nous envisageons également d'améliorer le monitoring et l'automatisation au sein du dépôt GitLab. L'objectif serait d'intégrer davantage d'indicateurs de suivi (tests, qualité de code, couverture, alertes...) directement dans les pipelines CI/CD, et de pousser l'automatisation sur des

tâches comme les déploiements, les tests d'intégration ou le linting, afin de gagner en efficacité et en réactivité tout au long du cycle de développement.

 Edit this page



> plan-du-rapport

> Conclusion

Conclusion

En ce qui concerne les perspectives, bien que nous n'ayons pas eu le temps de finaliser le déploiement sur Kubernetes et d'intégrer ArgoCD, ces étapes sont prévues pour la suite du projet.

L'idée serait de mettre en place Kubernetes pour l'orchestration des conteneurs, puis d'automatiser les déploiements via ArgoCD, en veillant à rendre le processus de déploiement aussi fluide que possible. Nous avons également l'intention d'intégrer des outils de monitoring pour optimiser la gestion des performances en production.

Cependant, plusieurs défis restent à relever avant d'atteindre cette étape. L'un des principaux obstacles est la configuration et l'intégration de ces outils, notamment Kubernetes et ArgoCD, qui nécessitent une maîtrise plus approfondie de la gestion des ressources cloud et des processus d'automatisation.

Cela pourrait demander des ajustements supplémentaires pour garantir que notre solution est stable et évolutive à long terme.

"Pour les profs pressés qui veulent briller sans transpirer"

 Dans cette section, nous avons compilé pour vous — chers correcteurs surchargés — toutes les zones cliquables du jeu, disponibles en annexe. L'objectif ? Vous permettre de naviguer rapidement jusqu'à la fin de l'escape game sans avoir à chercher pendant des heures (comme vos étudiants). C'est une sorte de fast pass pédagogique pour profiter d'une démo complète sans perdre votre temps à cliquer partout. Parfait pour évaluer efficacement... ou juste briller en réunion.



 Edit this page