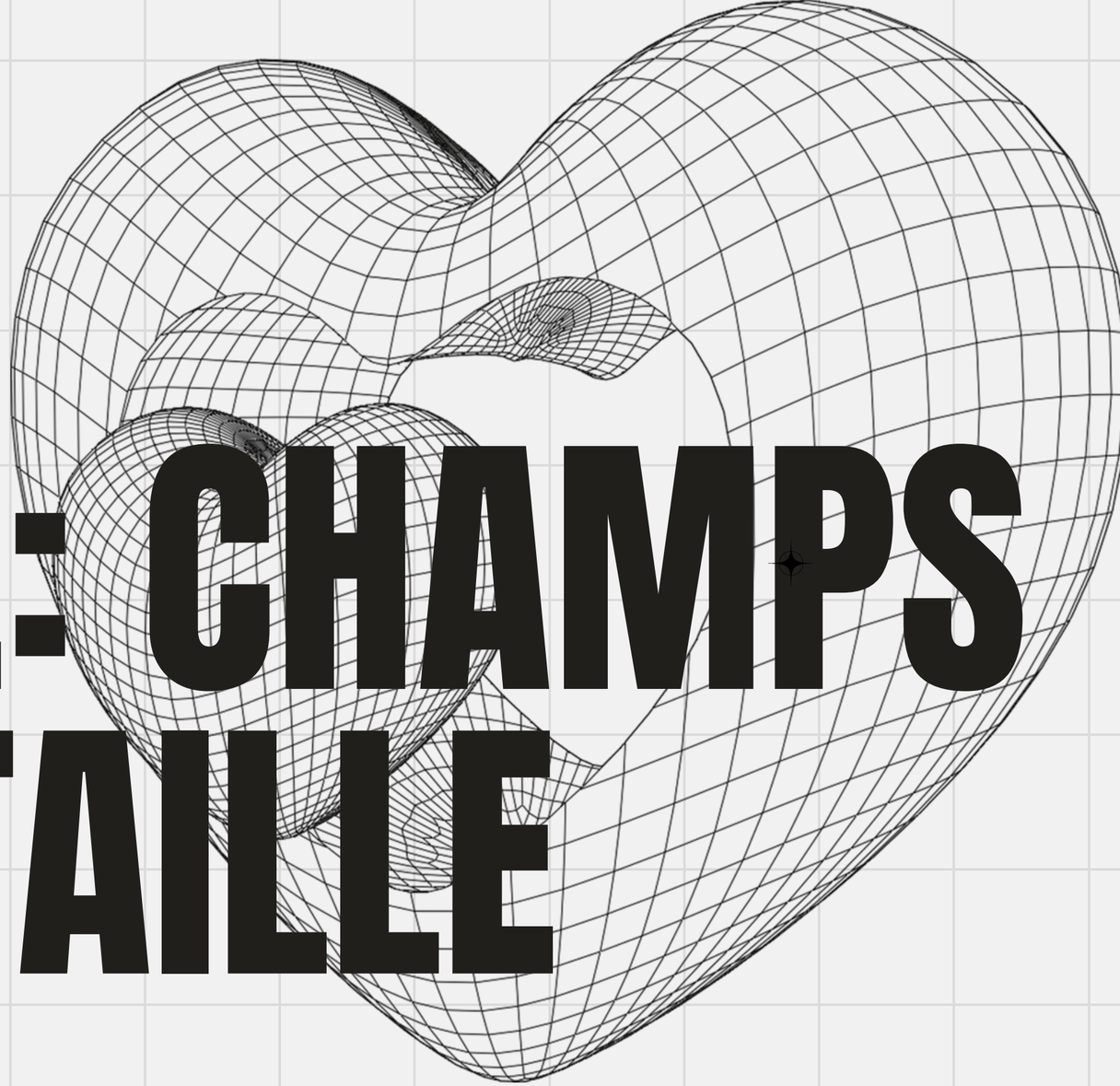
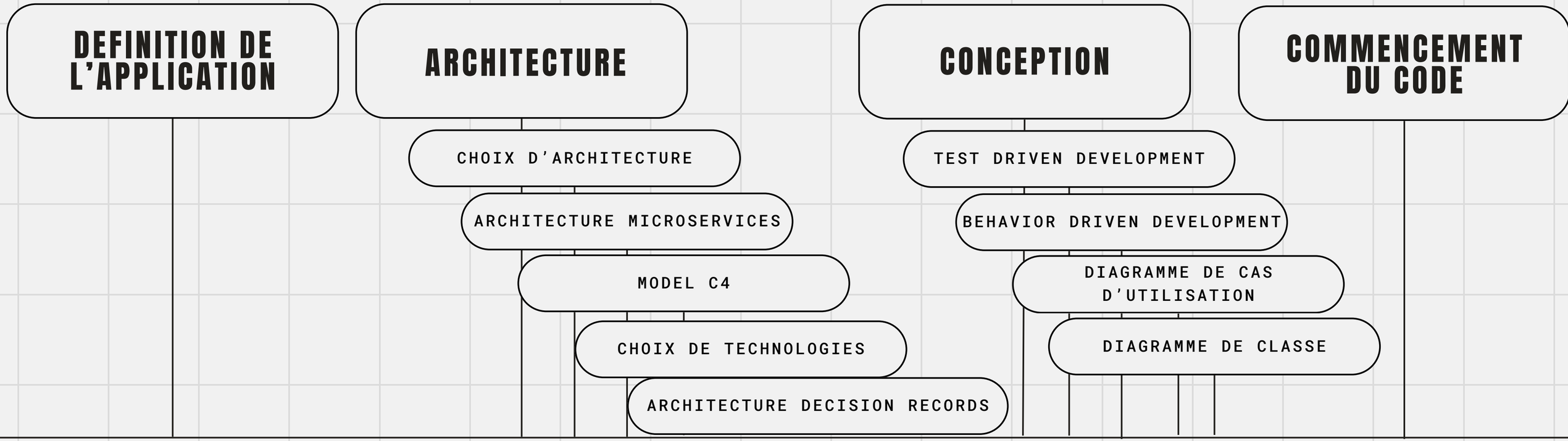
HEARSTONE- CHAMPS DE BATAILLE



SOFRONIEV MIHAIL
HAYET DERDOUR
MERIAM BEN HAJAL

TABLE OF CONTENT

FRISE CHRONOLOGIQUE



DEFINITION DE L'APPLICATION

DEFINITION DE L'APPLICATION

HEARTHSTONE: CHAMPS DE BATAILLE

CHAMPS DE BATAILLE

8 JOUEUR

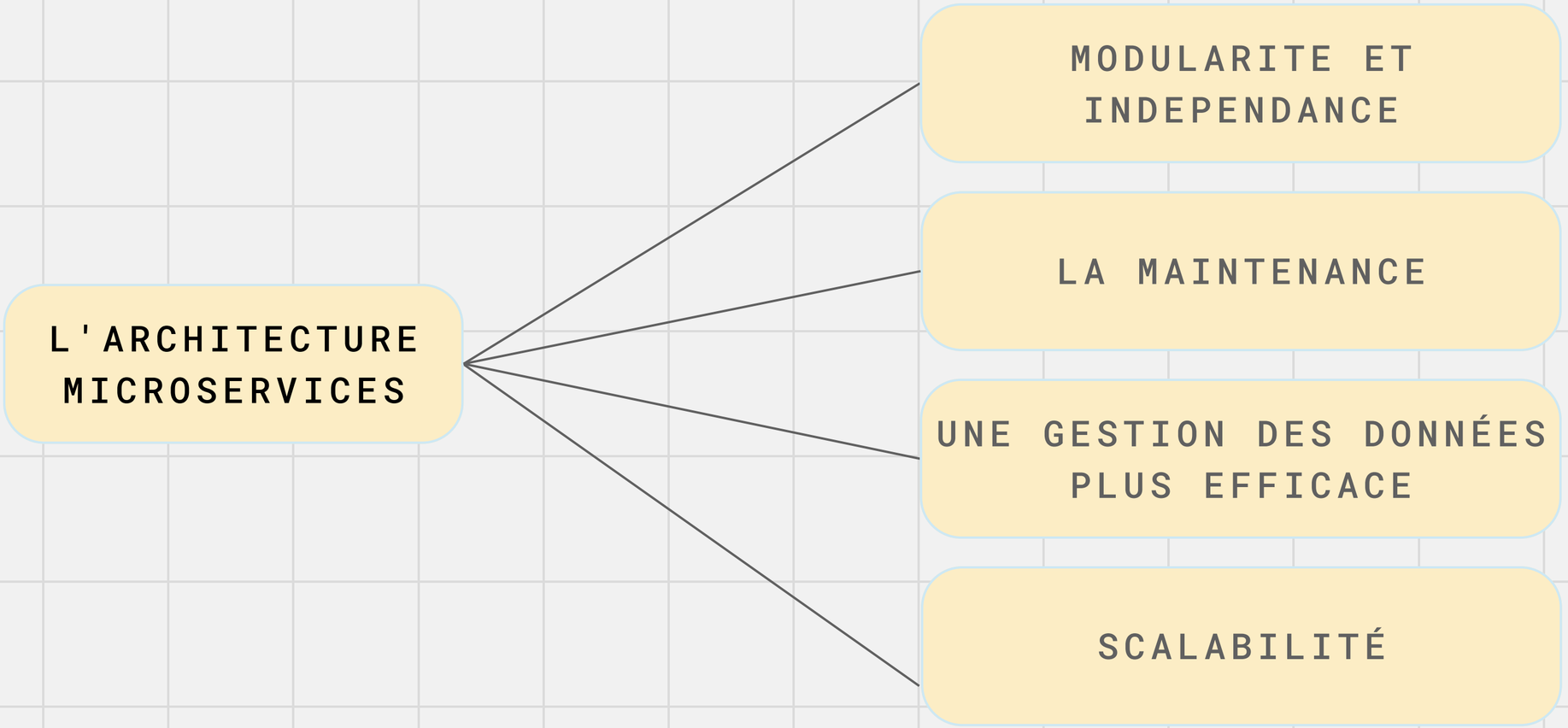
SERVITEURS

HÉROS

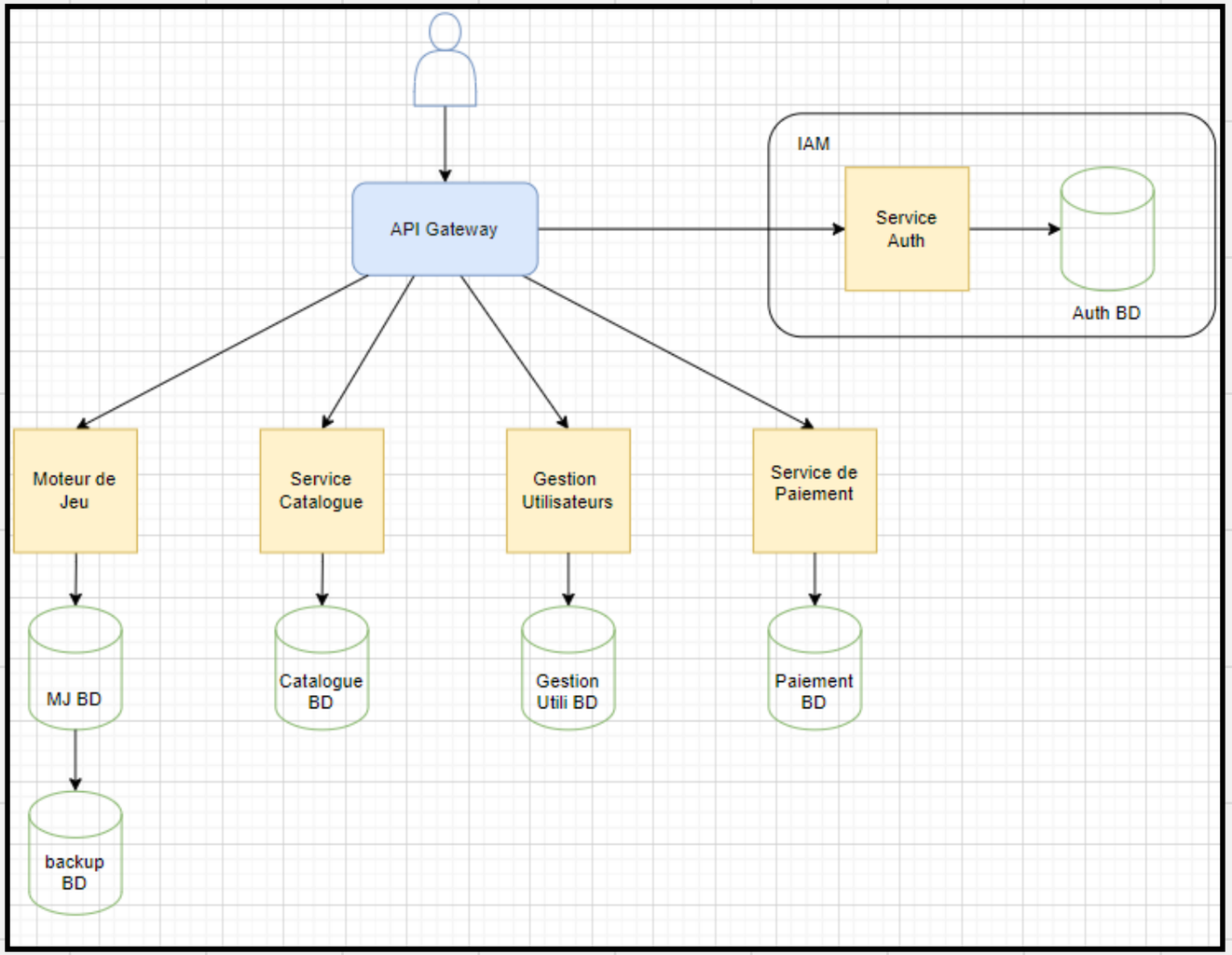
ARCHITECTURE

1. CHOIX D’ARCHITECTURE
2. ARCHITECTURE MICROSERVICES
3. MODEL C4
4. CHOIX DE TECHNOLOGIES
5. ARCHITECTURE DECISION RECORDS

CHOIX D’ARCHITECTURE

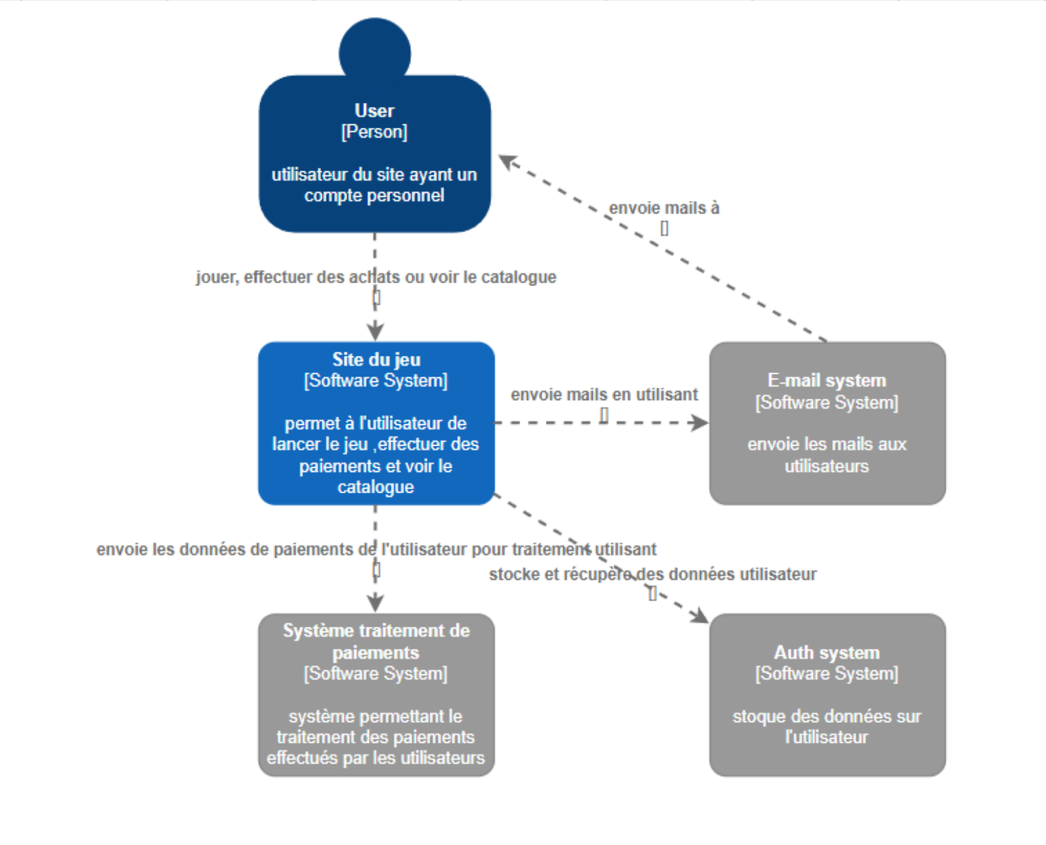


ARCHITECTURE MICROSERVICES



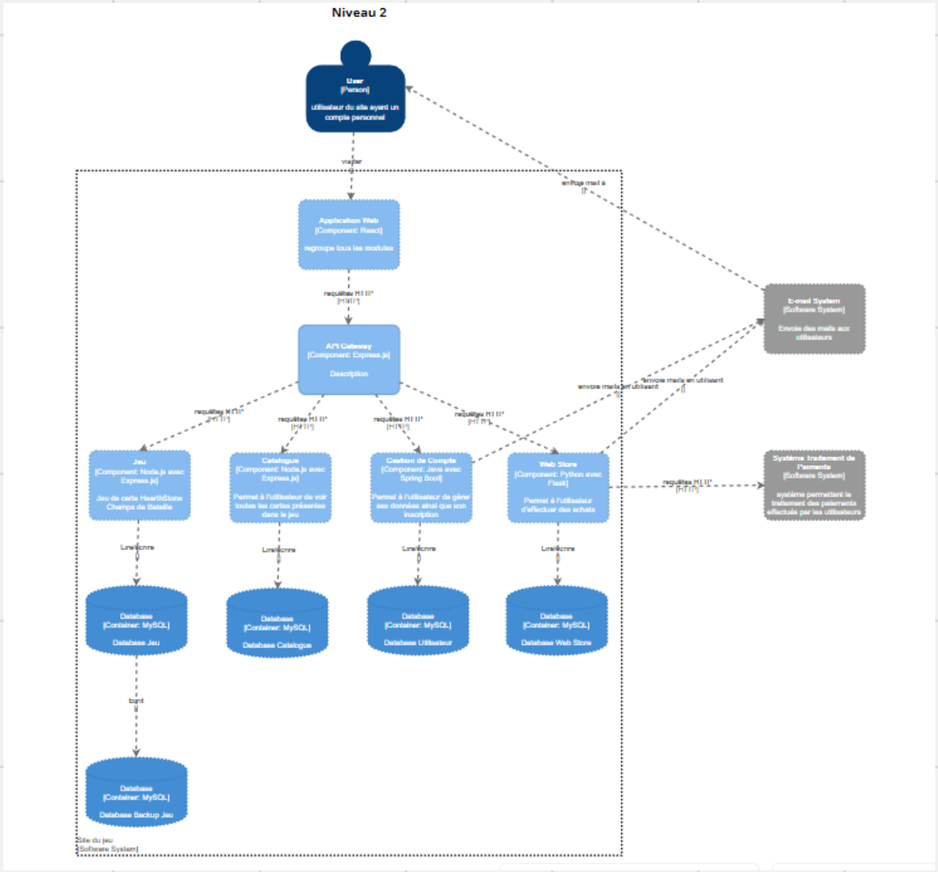
MODEL C4

Context Diagram



MODEL C4

Container Diagram



CHOIX DE TECHNOLOGIES

	INTERFACE UTILISATEUR	API	BD
SERVICE JEU	REACT	NODE.JS AVEC EXPRESS.JS	POSTGRESQL
SERVICE UTILISATEUR	REACT	JAVA AVEC SPRING BOOT	MYSQL
SERVICE CATALOGUE	REACT	NODE.JS AVEC EXPRESS.JS	MYSQL
SERVICE WEB STORE	REACT	PYTHON AVEC FLASK	MYSQL

ARCHITECTURE DECISION RECORDS

Titre : Utilisation de Node.js avec Express.js pour le service jeu

Date : 12/14/2023

Statut : Accepté

Contexte : On cherche à développer Hearthstone: Champs de bataille qui est un jeu stratégique à 8 joueurs dans l'univers de Hearthstone. Les joueurs recrutent des serviteurs, composent un plateau évolutif, et s'affrontent automatiquement dans des duels pour être le dernier héros en vie.

Décision : Nous avons décidé d'utiliser Node.js avec Express.js pour le développement des fonctionnalités Backend de notre jeu.

Conséquences :

- Positives :
  - Node.js est conçu pour être non bloquant et asynchrone, ce qui le rend bien adapté pour gérer un grand nombre de connexions simultanées. Dans un service de jeu, où la communication en temps réel et la réactivité sont cruciales, la nature événementielle et non bloquante de Node.js peut être avantageuse.
  - Node.js est souvent privilégié pour les applications nécessitant des fonctionnalités en temps réel, telles que les jeux multijoueurs. La capacité à gérer les connexions WebSocket et à transmettre des données en temps réel peut constituer un avantage significatif dans un scénario de jeu.
  - Node.js et Express.js sont réputés pour leur simplicité et leur légèreté. Le processus de développement peut être rapide, et le framework fournit juste assez de structure pour construire des applications évolutives sans introduire de complexité inutile.
  - L'écosystème Node.js dispose d'une vaste collection de modules (paquets npm) qui peuvent être facilement intégrés dans une application Express.js. Cela peut accélérer le développement en tirant parti de solutions existantes pour diverses fonctionnalités.
  - Node.js bénéficie d'une communauté nombreuse et active, et de nombreuses ressources d'apprentissage et tutoriels sont disponibles. Si votre équipe de développement est déjà familière avec JavaScript et Node.js, cela pourrait contribuer à un processus de développement plus fluide.
- Négatives :
  - Node.js est monoprocessus et basé sur des événements, ce qui pourrait limiter ses performances pour les tâches intensives en CPU. Les services de jeu impliquent souvent des calculs et des simulations, et la nature monoprocessus de Node.js peut ne pas exploiter pleinement les processeurs multi-cœurs.
  - Le modèle de programmation asynchrone de Node.js repose fortement sur les rappels (callbacks), ce qui peut conduire à ce que l'on appelle "l'enfer des rappels" (callback hell) - une situation où le code devient profondément imbriqué et plus difficile à lire. Gérer correctement le code asynchrone nécessite une bonne compréhension des motifs de rappels, des Promesses (Promises), ou de async/await, ce qui peut représenter une courbe d'apprentissage pour les développeurs.
  - Bien que Node.js excelle dans la gestion de nombreuses connexions simultanées, il pourrait ne pas exploiter pleinement la puissance des machines multi-cœurs. Cette limitation pourrait avoir un impact sur la scalabilité d'un service de jeu nécessitant une distribution efficace de charges de travail computationnelles importantes sur plusieurs cœurs.

Références :

Documentation officielle de node.js: https://docs.spring.io/spring-framework/reference/

Site officiel de express.js: https://expressjs.com/

COMMENT GÈRER LES DONNÉES

1. ENCRYPTAGE
2. HACHAGE DES PASSWORDS
3. DATA MASKING

ENCRYPTAGE

comment ?

HACHAGE DES PASSWORDS

comment ?

DATA MASKING

comment ?

SÉCURITÉ

- https
- encrypter les données
- utiliser des tokens pour l’authentification comme JWT
- respect du RGPD (règlement général sur la protection des données)
- serveurs de backup

CONCEPTION

1. TEST DRIVEN DEVELOPMENT
2. BEHAVIOR DRIVEN DEVELOPMENT
3. DIAGRAMME DE CAS D’UTILISATION
4. DIAGRAMME DE CLASSE

TEST DRIVEN DEVELOPMENT

1. TEST POUR VÉRIFIER QUE LES RÈGLES DU JEU SONT CORRECTEMENT IMPLÉMENTÉES DANS LE CODE

2. ECRIRE LE CODE MINIMUM POUR REUSSIR LE CODE

3. REFACTORISER LE CODE

BEHAVIOR DRIVEN DEVELOPMENT

UN UTILISATEUR SOUHAITE BÉNÉFICIER D'AVANTAGES EXCLUSIFS

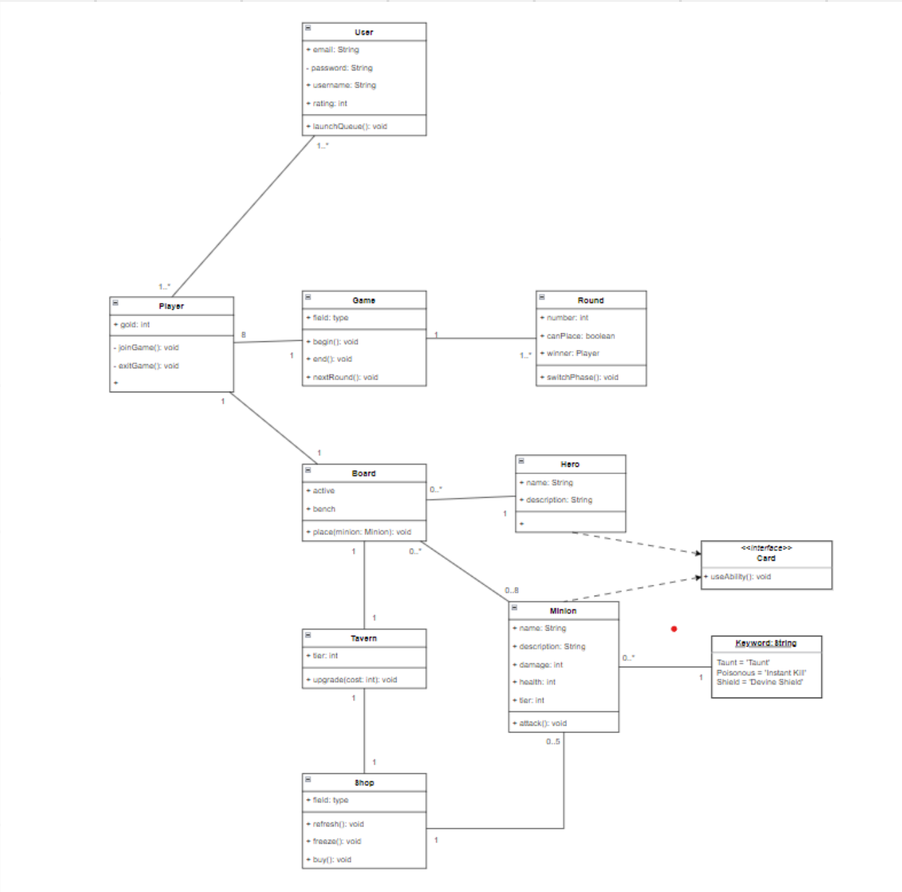
IL CHOISIT DE SOUSCRIRE À UN ABONNEMENT

SON ABONNEMENT EST ENREGISTRÉ, LUI DONNANT ACCÈS À DES FONCTIONNALITÉS SPÉCIALES ET À DES AVANTAGES EXCLUSIFS.

DIAGRAMME DE CAS D’UTILISATION

schema diagramme de cas d’utilisation

DIAGRAMME DE CLASSE



MERCI POUR VOTRE ATTENTION