1. Ce que l'équipe va faire et comment

Activités

Développement des microservices

Implémentation des 6 microservices avec leurs technologies respectives :

- auth-service
- user-service
- messaging-service
- media-service
- notification-service
- moderation-service

Mise en place du chiffrement

Implémentation du chiffrement bout-en-bout conforme au protocole Signal

Développement du modèle de modération

Création et entraînement d'un modèle de classification pour la mdération des contenus médias

Développement de l'application mobile

Création d'une application mobile respectant les normes d'accessibilité et de sécurité.

Tests

Tests unitaires, tests d'intégration, tests de charge et tests de sécurité

Déploiement

Configuration du cluster Kubernetes sur GCP et mise en place de l'infrastructure

Documentation

Documentation technique, guides d'utilisation et documentation API

Normes et bonnes pratiques

TypeScript

(auth-service, user-service, media-service)

- ESLint avec configuration stricte
- Interfaces bien définies
- Nommage en camelCase pour les variables et fonctions

Elixir

(messaging-service, notification-service)

- Formatage avec `mix format`
- Documentation avec ExDoc
- Respect des principes OTP
- Utilisation des structures de supervision adéquates

Python

(moderation-service)

- Respect de PEP 8
- Type hints obligatoires
- Documentation avec `docstrings`
- Utilisation du linter `ruff`

Rust

(modules WASM)

- Utilisation de `rustfmt`
- Documentation avec `rustdoc`
- Tests unitaires pour chaque module

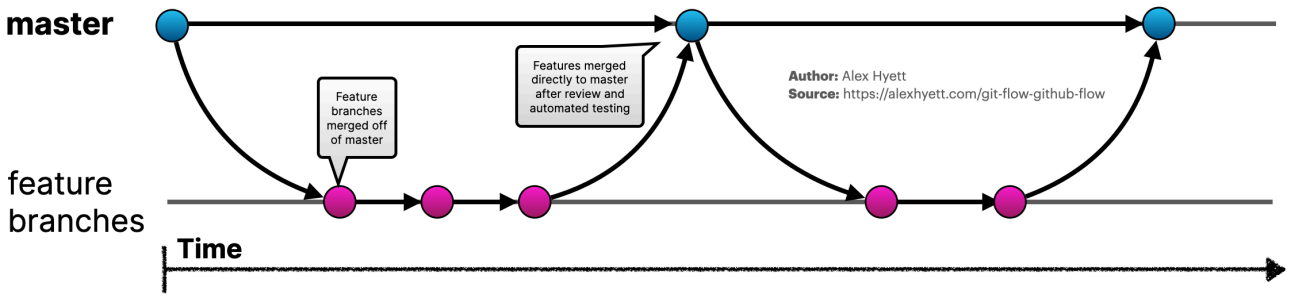
Norme de commits

Utilisation de conventional commits pour le nommage.

Stratégie de branches Git

- Convention des branches Git : `feature/`, `bugfix/`, `hotfix/`, `release/`
- Utilisation de Github flow pour une vériable intégration continue

GitHub Flow



2. Les rôles et responsabilités

Qui fait quoi ?

- **Agnes** : Chef de projet, responsable de l'architecture générale et de la coordination des équipes
- **Amin** : Développeur Fullstack (TypeScript) et DevOps
- **Dimitri** : Développeur (Fullstack) spécialisé dans la messagerie
- **David** : Responsable sécurité, Développeur Backend (Rust et TypeScript)
- **Zeyu** : Développeur Fullstack (TypeScript), Développeur IA/Data (Python)
- **Nathan** : Développeur IA/Data (Python) et Fullstack (TypeScript)
- **Tudy** : Développeur Backend (Elixir) et DevSecOps
- **Maia** : Développeuse IA/Data (Python) / ML Engineer
- **Imane** : Développeuse Frontend mobile, implémente également les fonctions sécurisées et le chiffrements dans l'application.
- **Gabriel** : Développeur Backend (Elixir), DevOps sur le cluster Kubernetes

Prérogatives

- **Décisions techniques** :
  - Architecture système : Agnes et Gabriel
  - Choix d'implémentation par service : le responsable du service
  - Standards de sécurité : David et Tudy
- **Validation des livrables** :
  - Code review : au moins 2 développeurs, dont le responsable du service concerné.
  - Tests de sécurité : validés par David ou Tudy
  - Qualité du code : validée via SonarQube
  - Validation finale : Agnes (chef de projet)

3. Le processus de développement

Méthode utilisée

Nous utiliserons une adaptation de la méthodologie Agile Scrum plus flexible pour ce projet étant donné que nous ne sommes présent à l'école 2 jours par semaine et avons des ressources limitées (pas de Scrum master) :

- **Sprints** de 2 semaines
- **Daily stand-up** meetings de 15 minutes
- **Sprint planning** au début de chaque sprint
- **Sprint review** et **retrospective** à la fin de chaque sprint
- **Product backlog** géré dans Jira, issu des user stories définies
- **Continuous Integration/Continuous Deployment** (CI/CD) pour l'intégration et le déploiement continus

Cette méthode a été choisie pour sa flexibilité et sa capacité à s'adapter aux évolutions du projet, particulièrement importante pour le développement du modèle de modération qui nécessitera des ajustements itératifs.

4. Les outils

Outils de configuration

- **Git** pour la gestion du code source
- **GitHub** pour l'hébergement des dépôts
- **GitHub Actions** pour l'automatisation des workflows CI/CD
- **SonarQube** pour le controle de la qualité et la sécurité du code
- **Docker** pour la conteneurisation des services
- **ArgoCD** pour le déploiement continu en GitOps
- **Kubernetes** (GKE) pour l'orchestration des conteneurs
- **Terraform** pour l'Infrastructure as Code sur GCP

Gestion des tickets

- **Jira** pour la gestion du backlog, des sprints et des bugs
- **Confluence** pour la documentation collaborative
- Organisation du backlog selon la WBS établie, avec hiérarchisation des tickets par importance
- Chaque ticket inclura des critères d'acceptation clairs

Tests

Tests unitaires

- TypeScript : Jest
- Elixir : ExUnit
- Python : Pytest
- Couverture de tests minimale : 70%

Tests d'intégration

- Tests entre microservices
- Tests de bout en bout

Tests de charge

- k6 pour simuler la charge utilisateur
- Tests de performance sur le messaging-service

Tests de sécurité

- OWASP ZAP pour les tests de vulnérabilité
- Audits de sécurité manuels

Qualité du code

- SonarQube pour l'analyse de code statique
- Conformité aux standards définis

5. La gestion des livraisons

Comment nous allons livrer

- Déploiement continu via GitHub Actions vers l'environnement de développement
- Déploiement hebdomadaire en environnement de staging
- Déploiement en production à la fin de chaque sprint réussi
- Processus de déploiement :
  1. Build et tests automatisés
  2. Analyse de code via SonarQube
  3. Déploiement automatique vers l'environnement cible
  4. Tests post-déploiement

Critères de validation

Pour qu'une fonctionnalité soit considérée comme livrée, elle doit satisfaire les critères suivants :

- Tous les tests unitaires et d'intégration passent
- Couverture de code ≥ 70%
- Aucun bug critique ou bloquant
- Aucune vulnérabilité de sécurité détectée
- Code review approuvée par au moins 2 membres de l'équipe
- Documentation mise à jour dans Confluence
- Conforme aux exigences non fonctionnelles (performance, sécurité)
- Pour les composants de chiffrement et sécurité : validation obligatoire par David ou Tudy
- Pour les fonctionnalités du modèle de modération : métriques de précision ≥ 85%

6. Monitoring et observabilité

Monitoring du cluster Kubernetes

Outils de monitoring

- Google Cloud Monitoring pour la surveillance des métriques du cluster GKE
- Prometheus pour la collecte des métriques techniques
- Grafana pour la visualisation des tableaux de bord
- Loki pour la gestion des logs

Alertes

- Alertes automatiques configurées pour l'utilisation des ressources (CPU, mémoire)
- Alertes sur la latence des services et le taux d'erreur
- Notification via et email en cas d'incident

Gestion des logs

- Centralisation des logs avec Google Cloud Logging
- Rétention des logs : 30 jours pour les logs standards, 90 jours pour les logs de sécurité
- Format de logs standardisé pour tous les microservices
- Traçabilité des requêtes avec des identifiants de corrélation entre services

7. Documentation

Documentation des APIs

Spécification

- Utilisation de OpenAPI/Swagger pour documenter toutes les APIs REST
- Utilisation de Protocol Buffers pour documenter les interfaces gRPC
- Versionnement des APIs intégré à la documentation

Processus

- Documentation générée à partir du code quand possible
- Documentation mise à jour à chaque modification d'API
- Revue de la documentation incluse dans le processus de code review

Publication

- Documentation centralisée dans Confluence
- Documentation API interactive accessible via un endpoint dédié (/api/docs)
- Exemples de requêtes et réponses pour chaque endpoint

Documentation technique

- Architecture détaillée sur Confluence avec diagrammes
- Guide d'installation et de déploiement
- Procédures opérationnelles (backup, restauration, scaling)
- Guide de contribution pour les développeurs
- Documentation du processus de chiffrement et de la gestion des certificats

8. Mesures et métriques de qualité

Métriques techniques

- Couverture de code : minimum 70%
- Dette technique : suivi via SonarQube, limiter à max 5% du code
- Temps de réponse des API : < 200ms pour 90% des requêtes
- Précision du modèle de modération : > 85%
- Uptime des services : > 99%
- Temps moyen de détection d'incident (MTTD) : < 5 minutes
- Temps moyen de résolution d'incident (MTTR) : < 2 heures

Revues et audits

- Revue de code pour chaque pull request
- Audit de sécurité mensuel
- Revue d'architecture mensuelle
- Audit de performance avant chaque mise en production
- Revue de la documentation des APIs à chaque fin de sprint