perplexity

Y a t'il quelque chose d'autres que l'on peut ajouter ?

Faire un récapitulatif exhaustif de la structure en allant dans les détails, faire un récapitulatif de ce qu'il y a à faire en prenant les example des projets existants. Faire un plan de travail.

Voici un **récapitulatif exhaustif** de la structure recommandée pour ton starter kit microservices Java/Maven, enrichi de suggestions additionnelles, d'une synthèse des tâches à réaliser (en s'appuyant sur des exemples de projets existants), et d'un plan de travail détaillé.

1. Éléments à ajouter pour un starter kit vraiment complet

- **Tests automatisés** (unitaires, intégration, end-to-end) avec exemples pour chaque microservice.
- **Gestion centralisée des configurations** (Spring Cloud Config, Vault, Consul).
- **Gestion des secrets** (HashiCorp Vault, ou intégration cloud équivalente).
- **Gestion des versions et CI/CD** (GitHub Actions, GitLab CI, Jenkins, avec pipelines préconfigurés).
- **Documentation générée** (Swagger/OpenAPI pour les APIs, AsciiDoc/Markdown pour la doc technique).
- **Gestion avancée des logs** (ELK Stack ou Loki, avec dashboards Grafana).
- **Support du multilingue et de l'internationalisation** pour les frontends.
- **Gestion des événements et de la communication asynchrone** (Kafka, RabbitMQ, ou autre broker).
- **Sécurité avancée** (scans de vulnérabilités, SAST/DAST, dépendances à jour).
- **Exemples de scripts d'initialisation de données** (liquibase/flyway).
- **Support de la conteneurisation** (Dockerfiles, docker-compose, Helm charts pour Kubernetes).
- **Support de la gestion fine des accès** (RBAC, ABAC, policies Keycloak).
- **Exemples de connecteurs cloud (AWS, Azure, GCP)** et IoT (MQTT, CoAP).
- **Module de supervision côté client** (intégration d'un agent open source, activation à la demande).
- **Gestion dynamique de la configuration** (reload sans redémarrage)[1].
- **Personnalisation de l'arborescence et des templates** (fichiers de config dynamiques, adaptation à chaque microservice)[2].

2. Structure détaillée recommandée

```
starterkit-parent/
├── pom.xml
├── microservices/
│   ├── service-licence/
│   │   ├── domain/
│   │   ├── application/
│   │   ├── infrastructure/
│   │   ├── adapters/
│   │   ├── bootstrap/
│   │   ├── test/
│   │   └── pom.xml
│   ├── service-droits/
│   │   └── ... (même structure)
│   ├── service-X/
│   │   └── ...
│   └── common-lib/
│       └── ...
├── api-gateway/
│   ├── src/
│   ├── config/
│   └── pom.xml
├── config-server/
│   ├── src/
│   ├── config-repo/
│   └── pom.xml
├── metrics-observability/
│   ├── prometheus/
│   ├── grafana/
│   ├── otel-collector/
│   ├── jaeger/
│   └── docker-compose.yml
├── iot-gateway/
│   ├── src/
│   ├── mqtt-adapter/
│   └── pom.xml
├── remote-support/
│   ├── java-remote-desktop/
│   ├── api-support/
│   └── pom.xml
├── frontends/
│   ├── web-app/
│   ├── mobile-app/
│   └── iot-dashboard/
├── deployment/
│   ├── docker/
│   ├── k8s/
│   ├── terraform/
│   └── ansible/
├── docs/
│   ├── architecture.md
│   ├── api-docs/
│   └── how-to/
├── .github/
```

```
│   └── workflows/
└── README.md
```

3. Synthèse des tâches à réaliser (inspiré des projets existants)

a. Initialisation du projet

- Créer un archetype Maven ou un générateur CLI (inspiré de JHipster, Spring Initializr, Cookiecutter).
- Définir un format de description (YAML/JSON) pour la génération automatisée.

b. Microservices Java

- Générer la structure DDD/Hexa pour chaque service (domain, application, infrastructure, adapters).
- Ajouter des exemples de tests unitaires et d'intégration[3].
- Intégrer Spring Boot, Spring Data, Spring Security.

c. Gestion des droits

- Intégrer Keycloak (configuration, scripts d'initialisation, policies).
- Ajouter un module d'intégration Keycloak dans chaque microservice.

d. Metrics & Observabilité

- Ajouter Prometheus, OpenTelemetry, Grafana, Jaeger/SigNoz.
- Générer les endpoints d'export de metrics dans chaque microservice.

e. API Gateway & Config Server

- Générer un API Gateway (Spring Cloud Gateway, Traefik).
- Ajouter un config server (Spring Cloud Config) avec repo Git local ou distant[1].

f. IoT & Cloud

- Générer un module iot-gateway (exemple MQTT, CoAP).
- Préparer des connecteurs cloud (AWS, Azure, GCP) et des scripts Terraform/Ansible pour le déploiement[4].

g. Supervision côté client

- Intégrer un agent open source Java de prise en main/supervision (jrDesktop ou équivalent).
- Ajouter une API pour déclencher et monitorer les sessions de support.

h. Frontends

- Générer des modules pour web-app, mobile-app, et dashboards IoT, chacun avec un point d'entrée API.

i. CI/CD & Documentation

- Ajouter des pipelines CI/CD (GitHub Actions, GitLab CI).
- Générer la documentation API (Swagger/OpenAPI).
- Générer la documentation technique et d'exploitation.

j. Logs, Sécurité, Tests

- Intégrer ELK/Loki pour les logs.
- Ajouter des scans de sécurité (OWASP, Snyk, etc.).
- Générer des scripts de tests automatisés pour chaque module[3].

4. Plan de travail (macro-étapes)

Étape	Description	Durée estimée
1	Rédaction du cahier des charges détaillé, choix des outils et technologies	1-2 semaines
2	Développement du générateur de structure (archetype/CLI)	2-3 semaines
3	Génération des microservices DDD/Hexa, intégration des dépendances de base	2-3 semaines
4	Intégration IAM (Keycloak), metrics (Prometheus, Grafana), logs (ELK), config server	2-3 semaines
5	Modules IoT, connecteurs cloud, supervision client	2-4 semaines
6	Génération des frontends, documentation, pipelines CI/CD	2-3 semaines
7	Tests, validation, itérations, documentation finale	2-3 semaines

5. Suggestions additionnelles

- **Blueprints personnalisables** : Permettre à l'utilisateur de définir des "presets" pour différents types de microservices (REST, event-driven, IoT, etc.).
- **Plug-ins/extensibilité** : Prévoir un système de plugins pour intégrer facilement de nouveaux connecteurs ou briques techniques.
- **Interface web de génération** : Offrir une interface web (façon Spring Initializr ou JHipster) pour générer le projet à la demande.
- **Exemples d'intégration avec des outils de gestion de configuration avancée** (Vault, Consul)[1].
- **Support multi-environnement** (dev, staging, prod) avec des fichiers de configuration adaptés.

En résumé :
Un starter kit microservices avancé doit aller bien au-delà d'une simple arborescence : il doit

intégrer tests, CI/CD, gestion centralisée de la config, IAM, metrics, logs, documentation, connecteurs cloud/IoT, supervision client, et être personnalisable.
Le plan de travail ci-dessus, inspiré des meilleurs projets open source existants, permet de structurer le développement de façon progressive et modulaire.

⁂

1. programming.configuration_management
2. programming.file_system_customization
3. tools.test_development
4. tools.infrastructure_as_code