Phase 2 - BrokerX

Contexte

Vous poursuivez le rôle d'architecte logiciel pour BrokerX+, une plateforme de courtage en ligne destinée aux investisseurs particuliers. Après la phase 1 (monolithique évolutif), l'organisation vise maintenant : (i) l'exposition sécurisée d'une API RESTful, (ii) l'industrialisation de l'observabilité et de la performance (tests de charge, load balancing, caching), et (iii) l'évolution vers une architecture microservices orchestrée par une API Gateway, tout en respectant les exigences fonctionnelles et non-fonctionnelles du cahier de charge (ordres, portefeuilles, données de marché, conformité, audit).

Votre mandat consiste à planifier et exécuter la transition en gardant la traçabilité (Arc42, 4+1, ADR) et en démontrant des gains mesurés sur la latence, le débit, la disponibilité et l'observabilité (4 Golden Signals).

Objectifs d'apprentissage

À l'issue de ce projet, l'étudiant·e devra être capable de :

- Exposer une API RESTful conforme aux bonnes pratiques, documentée (OpenAPI/Swagger), sécurisée (CORS, auth de base/JWT).
- Mettre en place l'observabilité (logs structurés, métriques Prometheus, dashboards Grafana) et mesurer les 4 Golden Signals sous charge réaliste (k6/JMeter).
- Optimiser la performance par load balancing (NGINX/HAProxy/Traefik) et caching (mémoire/Redis) sur endpoints critiques.
- Migrer vers des microservices et publier via une API Gateway (Kong/KrakenD/Spring Cloud Gateway), avec routage, sécurité et comparaisons A/B (direct vs gateway).
- Respecter le cahier de charge : domaines (Comptes & Clients, Ordres & Appariement, Données de marché, Portefeuilles), NFR (latence P95, throughput, disponibilité) et exigences de conformité/audit.
- Maintenir la documentation architecturale (Arc42 + 4+1) et consigner les décisions (ADRs).

Tâches à réaliser (livrables & critères d'acceptation)

1) Analyse métier & DDD - Pilotée par le cahier de charge

Clarifier le périmètre fonctionnel prioritaire :

≥ 5 UC du cahier entièrement décrits et implements ;

2) Architecture & décisions — Monolithique → REST

Concevoir/ajouter la couche API RESTful sur le noyau métier (Hexagonal ou MVC), routes versionnées, codes HTTP, erreurs normalisées.
Documenter l'architecture (4+1 + Arc42) :

- 4+1 : Logique, Processus (C&C), Déploiement, Développement, Scénarios.
- Arc42 : §§ 1–8 (contexte, contraintes, concept solution, décisions, qualité, risques).
  ADRs (≥ 3) : style architectural, persistence/transactions, stratégie d'erreurs/versionnage, conformité & audit (exactly-once/effectively-once).
  CA : dépendances dirigées, séparation domain↔infra, routes cohérentes, Swagger publié.

3) Persistance & intégrité

Schéma & intégrité : modèle ER/UML, migrations reproductibles, seeds, transactions ; contraintes (unicité, FK, index).
Implémentation : DAO/Repository/ORM, tests d'intégration DB conteneurisée ; idempotency-key pour dépôts/ordres ; journal d'audit append-only.
CA : CRUD robustes sur agrégats clés (Comptes, Ordres, Positions), rollback sur erreurs.

4) Implémentation du prototype REST & sécurité minimale

Domaine & services applicatifs : règles pré-trade, normalisation & horodatage, appariement interne basique, notifications.
API REST : endpoints pour UC Must. Swagger + collection Postman.
Sécurité : CORS, auth simple (Basic/JWT), validation/assainissement entrées.
CA : scénario bout-en-bout démontrable (VM), erreurs normalisées JSON.

5) Observabilité, tests de charge, optimisation

Observabilité : logs structurés, métriques applicatives (Prometheus), dashboards Grafana (4 Golden Signals : latence P95/99, trafic RPS, erreurs 4xx/5xx, saturation CPU/RAM/threads).
Tests de charge (k6/JMeter/Artillery) : scénarios réalistes (consultation stocks/carnets, génération de rapports, mises à jour/ordres à cadence élevée), stress test progressif jusqu'au seuil.
Load balancing : NGINX/HAProxy/Traefik ; répéter les tests pour N = 1,2,3,4 instances ; graphiques comparatifs (X=instances, Y=latence/RPS/erreurs/saturation). Tolérance aux pannes (kill d'instance en charge).
Caching : endpoints coûteux (/api/stores/{id}/stock, /api/reports/sales, carnets/cotations) en cache mémoire ou Redis ; règles d'expiration/invalidation ; mesurer gains/risques (stale).
CA : dashboards + tableaux comparatifs avant/après (latence, RPS, erreurs, charge DB).

6) Migration Microservices & API Gateway

Découpage logique : extraire ≥ 3–4 services (Ordres/Matching, Reporting, Stock/Portefeuilles); un conteneur par service.
API Gateway (Kong/KrakenD/Spring Cloud Gateway) : routage dynamique, ajout d'en-têtes/clé API, CORS, (optionnels : journaux d'accès, throttling/quota). LB via Gateway entre ≥ 2 instances d'un microservice ; tester la distribution.
Comparaisons : Appels directs (ancienne archi) vs via Gateway (nouvelle) sous charge ; analyser latence, taux d'erreurs, traçabilité ; intégrer au dashboard.
CA : appels fonctionnels via Gateway, gains/impacts mesurés et argumentés.

7) CI/CD & conteneurisation

Conteneurs : Dockerfile multi-stage, `docker-compose.yml` (app/services + DB + Prometheus + Grafana + Gateway + seed), healthchecks `/health`.
CI : lint → build → tests (unit/int/E2E) → artefacts ; badge. CD : script de déploiement (compose/swarm/systemd) + rollback simple.
CA : pipeline déterministe < 10 min ; déploiement en une commande.

8) Documentation finale & démonstration

Arc42 + 4+1 à jour, ADRs consolidés (≥ 3), README (setup, jeu de tests, endpoints), runbook (opérations, observabilité, pannes). Rapport comparatif : monolithique REST → optimisé (LB/cache) → microservices+Gateway, avec NFR cibles du cahier (paliers latence/throughput/dispo) positionnées comme objectifs/planchers et résultats atteints.
CA : reproductibilité < 30 min sur VM ; liens/captures Grafana/Prometheus.

Définition de Fini (DoD)

- UC Must du cahier implémentés de bout-en-bout via l'API REST (domain + persistence + entrée) ; erreurs normalisées.
- Observabilité opérationnelle (logs, métriques Prometheus, dashboards Grafana) + campagnes de charge avec comparatifs (avant/après, N instances).
- LB et cache en production (compose) avec impacts chiffrés.
- API Gateway en place, microservices conteneurisés, routage vérifié, scénarios de test direct vs gateway comparés.
- 4+1 cohérent, Arc42 (sections cœur), ≥ 3 ADR approuvés ; CI/CD fonctionnels.
- Rapport PDF + dépôt projet (zip) remis (code, scripts, dashboards, doc).

Grille d'évaluation — Projet intégré BrokerX+

Critères	Pondération	Excellent (A)	Satisfaisant (B–C)	Insuffisant (D)	Non réalisé (F)

1. Analyse métier & DDD	15%	UC du cahier (≥ 5) bien détaillés, BCs/glossaire /modèle clairs & cohérents	UC partiels, modèle incomplet	UC superficiels/incohérents	Absence d'UC/modèle
2. API REST & Sécurité	15%	Endpoints complets, versionnés, erreurs normalisées, Swagger/Postman, CORS+Auth	Endpoints OK mais doc/sécurité partielles	API limitée ou fragile	Pas d'API
3. Persistance & Intégrité	15%	Schéma robuste, migrations/seed, idempotence, audit append-only	Design incomplet/tests minimes	CRUD fragile, peu de garanties	Pas de persistance
4. Observabilité & Charge	20%	4 Golden Signals, scénarios charge réalistes, comparatifs clairs & interprétés	Obs/tests partiels	Mesures peu fiables	Aucune mesure
5. LB & Caching	10%	LB multi-instances + cache efficaces, gains mesurés et discutés	LB/cache partiels	Gains non démontrés	Aucun LB/cache

6. Microservices & Gateway	15%	3–4 services + e-commerce extraits, Gateway opérationnelle (routage, règles), A/B direct vs gateway	Migration/gateway incomplètes	Microservices/Gateway peu fonctionnels	Aucun découpage/Gateway
7. Doc & Décisions (Arc42/4+1/ADR)	10%	Arc42+4+1 complets, ≥ 3 ADR solides, risques/qualité traités	Doc partielle, ADR limités	Doc superficielle	Aucune doc

Barème indicatif
A (85–100 %) : Projet complet, robuste, gains mesurés, doc claire.
B (70–84 %) : Projet solide mais lacunes (doc/tests/obs).
C (60–69 %) : Fonctionnel mais partiel, preuves de performance limitées.
D (50–59 %) : Incomplet, architecture fragile ou mal documentée.
F (<50 %) : Non fonctionnel ou livrables incohérents.