**FreshBazaar — Phase-Wise End-to-End Roadmap**

Notes: Each phase contains: Goals, Deliverables, Integration & Test steps, Deployment steps, and Why (rationale).

**Phase 0** — Project Setup & Policy (pre-dev, mandatory)

**Goal:** Prepare repository, standards, infra hooks and automation so every later step plugs into CI/CD, security and telemetry.

**Deliverables:**

* Monorepo skeleton (the industry-grade structure).
* Per-service build.gradle (or Maven POM) and root CI config placeholders.
* Coding standards, PR template, commit message policy.
* Branch strategy (master, dev, feature/\*).
* Secrets handling plan (HashiCorp Vault / AWS Secrets Manager).
* Basic infra-as-code repo skeleton (Terraform modules + README).
* Enable GitHub repo integrations (webhooks for CI, Sonar, Slack).

**Integration & Test:**

* Validate CI runs on simple test commits.
* Validate ability to read secrets from Vault/CI.

**Deployment:**

* No app deploy yet — just infra provisioning plans and CI hooks.

**Phase 1** — MVP: Core Features, Local & Staging Deployments

**Goal:** End-to-end working product demo: auth, trader, product listing, basic customer UI, deployed to staging.

**Core Features:**

* Auth-service (JWT, roles: TRADER, CUSTOMER).
* Trader-service (profile CRUD).
* Product-service (create/update daily rates, images can be local for now).
* Customer-service (browse products, search basic).
* Gateway-service (Spring Cloud Gateway) + simple API docs (OpenAPI).
* React web frontend (TypeScript + Vite) — login, product list, trader profile.
* DB: PostgreSQL (single instance) with Flyway migrations.
* Dockerize each service; infra/docker/docker-compose.yml for local & staging.
* Unit tests + basic integration tests for each service.
* CI pipeline: build, test, sonar cloud (or SonarQube) quality gate.

**Integration & Test:**

* Contract tests / OpenAPI validation between services.
* End-to-end tests (API → DB) in CI for each merge to master.
* Frontend integration tests (React Testing Library) running in CI.

**Deployment:**

* Deploy stack to a staging environment via Docker Compose / simple k8s manifests.
* Expose UI through Nginx or ingress.
* Confirm staging can be used for demos.

**Phase 2** — Quality, UX & Feature Enhancements

**Goal:** Make product production-ready for real users: improve UX, add reviews, contact/bid, image storage, and fast search.

**Features**

* Review-service (feedbacks: ratings + comments).
* Bidding stub (customers can place bids; stored in DB).
* File storage: move images to S3 / compatible object storage (or MinIO for infra parity).
* Add Elasticsearch for fast search & range/aggregation queries (price comparisons).
* Add Redis for caching hot reads (market dashboard).
* Bulk upload API (CSV) for big traders.
* Frontend: better UI/UX, filters, sorting, responsive layout.
* Mobile: PWA capability or start React Native skeleton (reusing APIs).
* API versioning and rate limiting in Gateway.

**Integration & Test:**

* Integration with S3, Elasticsearch and Redis in staging.
* Load tests for product search endpoints (k6 or Gatling).
* Security scans: SonarQube quality gates and Veracode static scan in CI.

**Deployment:**

* Move to Kubernetes (single cluster): Helm charts or kustomize under infra/k8s.
* CI pipeline: build images, push to registry (ECR/GCR/DockerHub), deploy to staging k8s.
* Observability: basic Prometheus + Grafana dashboards; logs to Splunk.

**Phase 3** — Scalability, Reliability & Production Harden

**Goal:** Production-grade infra: autoscaling, backups, monitoring, CI/CD gating, multi-env promotion.

**Features / Ops:**

* Production Kubernetes cluster (managed: EKS/GKE/AKS).
* Horizontal Pod Autoscaling + Cluster Autoscaler.
* Database hardening: RDS/Aurora or managed PostgreSQL with read replicas.
* Use CDN for static assets (CloudFront / Cloud CDN).
* Use Terraform fully for infra provisioning.
* Implement CI/CD promotion: develop → staging (auto), staging → main (manual promotion with approvals).
* Canary / blue-green deploy support in pipelines.
* Backups: DB automated backups + object storage lifecycle.
* IAM roles, network policies, WAF and baseline security posture.
* Splunk logging fully integrated; Prometheus alerting to Slack/pager.

**Integration & Test:**

* Chaos engineering smoke tests (optional) — e.g., simulate pod failures.
* DR drills for DB failover and restore.
* Penetration testing and Veracode policy enforcement for all services.

**Deployment:**

* CI runs full security and quality checks; protected branches for production only.
* Observability & SLOs in place (error budget, latency thresholds).

**Phase 4** — AI Add-Ons & Smart Features.

**Goal:** Add AI-driven features that create real value and a “wow” factor for recruiters.

**AI Features:**

* Price Predictor microservice (Python: FastAPI) — daily batch predictions pushed to Product-service.
* Market Trend service — trend visualizations and alerts.
* NLP Search service — accept natural queries, translate to filters, hit ES.
* Image classifier — auto-tag product images using a small CNN/transfer learning.
* Model registry & CI for models; schedule retraining pipelines (Airflow / GitHub Actions).

**Integration & Test:**

* ML model validation metrics stored and displayed in a model dashboard.
* Canary model rollout: A/B test model recommendations before full switch.
* Security review for model serving endpoints.

**Deployment:**

* Containerized AI services, deployed to k8s (separate namespace), use GPU nodes if necessary for heavy models.
* Data pipeline (Kafka or batch ETL) to feed historical price data.

**Phase 4** — Production Optimization & Commercialization.

**Goal:** finalize features for a product pitch (SLA, multi-region, billing, analytics).

**Features**

* Multi-region deployment (disaster recovery).
* Billing/subscriptions or enterprise onboarding flows.
* Advanced analytics for traders (sales, demand).
* GDPR / data compliance features (if applicable).
* Full docs, case study, demo video, public demo URL.

**Integration & Test**

* Multi-region failover testing.
* Final security audit and compliance checks.

**Deployment**

* Global CDN + latency optimization.
* Business continuity plan and runbooks.

# **Cross-Cutting Concerns (applies in all phases)**

* **Testing:** Unit tests, integration tests, contract tests, end-to-end tests.
* **Static Analysis:** SonarQube gates on PRs.
* **Security Scans:** Veracode scan step in CI before merge to main.
* **Secrets:** Vault/AWS Secrets Manager in all environments.
* **Monitoring:** Prometheus + Grafana + Splunk for logs.
* **Observability:** Structured logs (JSON), distributed tracing (OpenTelemetry / Jaeger).
* **Database Migrations:** Flyway for schema changes.
* **Feature Flags:** Unleash / LaunchDarkly for rollouts.

# **Recommendations — Tools & Technologies (concise)**

### **1) Build tool for Spring Boot**

**Recommendation:** **Gradle (Kotlin DSL)  
Why:** faster incremental builds, better multi-project builds (monorepo), modern dependency management, widely used in large teams.  
*(Maven is acceptable and stable — use it if you prefer convention-over-configuration.)*

### **2) Frontend: JavaScript or TypeScript?**

**Recommendation:** **TypeScript with React** (React + TypeScript + Vite)  
**Why:** Type safety reduces bugs, better DX at scale, highly recruiter-friendly. Use **Vite** for fast dev server & builds. Use **Redux Toolkit** or React Query for state/data fetching. For styling use Material UI or Tailwind.

### **3) Database choice (data + scalability)**

**Primary DB:** **PostgreSQL** (managed like AWS RDS / Aurora)  
**Search:** **Elasticsearch** (for market-wide search & aggregations)  
**Cache / Real-time:** **Redis** (caching, leaderboards, short-lived locks)  
**Object Storage:** **S3 / MinIO** for images  
**Why:** PostgreSQL is battle-tested for relational transactional data (products, bids, users). Elasticsearch handles analytical search and price aggregations. Redis improves latency for dashboards. Managed services (RDS, Elastic Cloud) reduce ops overhead.

### **4) Deployment**

**Recommendation:** **Docker → Kubernetes (managed: AWS EKS / GKE / AKS)** with GitHub Actions CI/CD.

**Why:** Containerized microservices + k8s autoscaling are industry standard and show recruiters systems & ops maturity. GitHub Actions integrates well with Sonar, Veracode, and pushing images to ECR/GCR.

**Additional:** Use **Terraform** for infra-as-code, **Helm** for k8s templating, **Flux/ArgoCD** if you want GitOps later.

### **5) DB Migrations**

**Recommendation:** **Flyway  
Why:** simple, transactional, easy integration with Spring Boot.

### **6) Testing & Code Quality**

* Java: JUnit 5 + Mockito + Spring Boot Test
* Frontend: Jest + React Testing Library + Playwright for E2E
* Code quality: SonarQube
* Security Scans: Veracode (or Snyk for dependency scanning)

### **7) Observability**

* **Metrics/Alerts:** Prometheus + Grafana
* **Tracing:** Jaeger / OpenTelemetry
* **Logs:** Splunk (user decided) or ELK stack