Security Design Document

Overview

The security design of the Belote system is grounded in the principle of **Security by Design** — integrating protection mechanisms into every stage of the software development lifecycle rather than treating them as an afterthought. The application consists of multiple distributed services (Server, Lobby Service, Game Service, Client), all communicating through a **secure, event-driven architecture** using **NATS**. Each component follows strict trust boundaries, validation, and communication rules to minimize attack surfaces and ensure data integrity.

1. Security Principles Applied

1.1 Least Privilege

Each service has access only to the resources and data it needs.

- The **Server** handles client requests but does not perform game logic.
- The **Game Service** and **Lobby Service** operate independently and cannot directly modify each other's internal states.
- Database credentials and environment variables are isolated per container and never shared globally.

1.2 Defense in Depth

Multiple layers of security ensure that no single failure compromises the entire system.

- Communication between services is brokered by **NATS**, which prevents direct exposure of internal APIs.
- Input validation occurs both at the **Server level** (from client inputs) and at the **Game Service level** (for internal consistency).
- Docker container isolation and controlled network access provide an additional layer of protection.

1.3 Zero Trust (Authoritative Server Model)

Clients are not trusted with any critical game logic. All game state transitions, move validations, and scoring occur exclusively on the **authoritative server side** (Game Service). Client inputs are treated as **unverified suggestions** that must pass server-side validation before affecting the system state. This prevents common multiplayer exploits such as speed hacks, replay attacks, or client data tampering.

1.4 Secure by Default

Every configuration, container, and dependency is designed to minimize risk without requiring user intervention.

- Default ports are secured via internal Docker networking rather than exposed publicly.
- No sensitive configuration values (e.g., credentials, API keys) are hard-coded or committed to version control.
- `.env` files are used locally, while **GitHub Secrets** and **Kubernetes Secrets** are planned for future deployment.

1.5 Auditability and Transparency

All communication and critical actions can be monitored and traced.
Prometheus metrics and container logs record system events, allowing detection of suspicious activity such as abnormal request spikes or failed authentication attempts.

2. Secure Communication and Data Handling

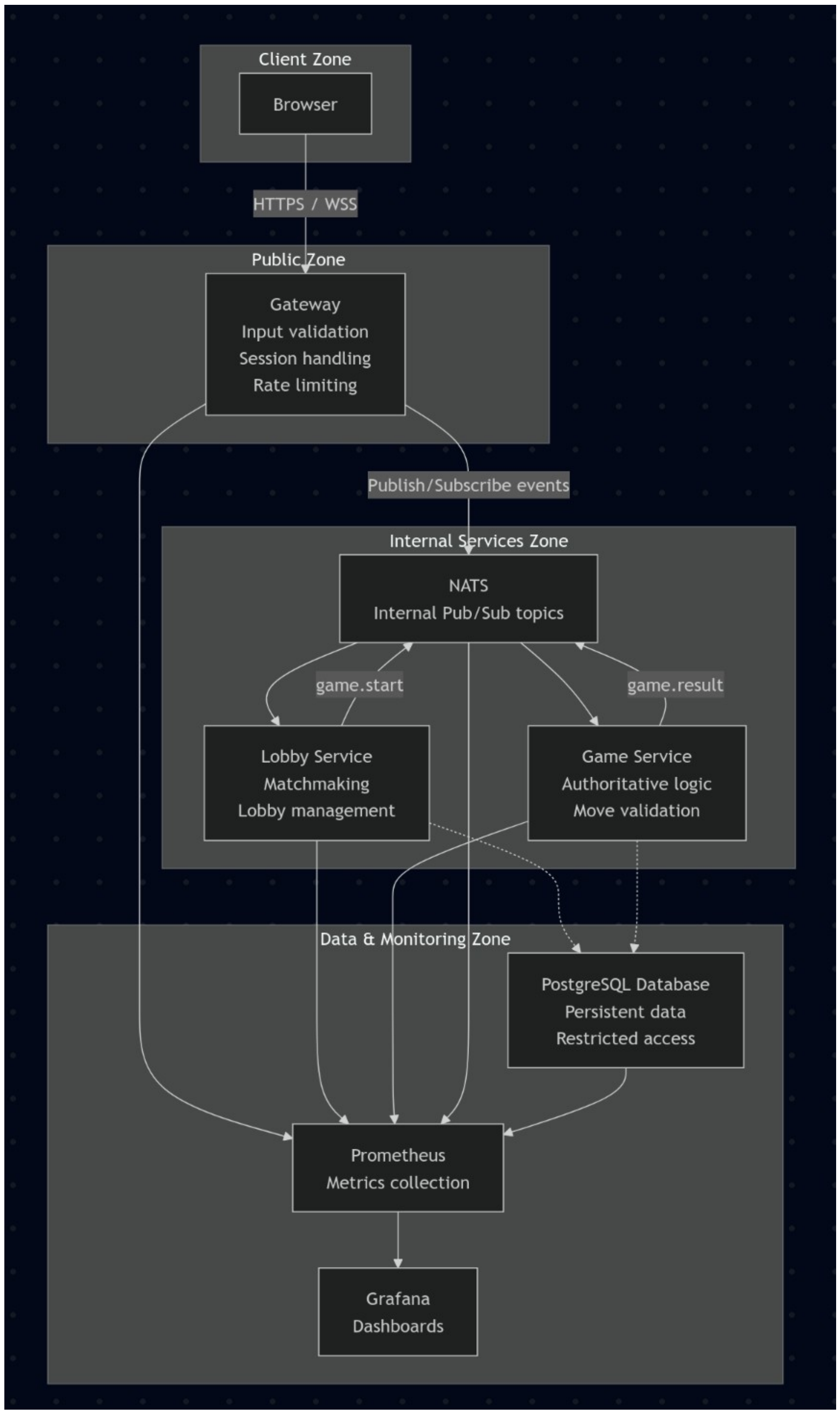
- **Internal Communication:**
  All microservices communicate exclusively via NATS using subjects not exposed to the public.
  This prevents direct HTTP access between internal services.
- **External Communication:**
  Client-to-server connections use **WebSockets** or HTTPS. All sensitive payloads (e.g., player actions) are validated against session tokens before processing.
- **Data Storage:**
  The PostgreSQL database stores only essential, non-sensitive game data.
  Player credentials will be stored in a dedicated **Authentication Service** planned for future development, which will handle encryption and session management using **JWT tokens** and **bcrypt** password hashing.
- **Logs and Metrics:**
  Logs avoid storing personally identifiable information (PII). Prometheus metrics are used for performance tracking, not for sensitive data.

3. Future Security Enhancements

To further strengthen the system's security posture, the following steps are planned:

- **Authentication & Authorization Service:**
  A dedicated microservice using JWT tokens and role-based access control (e.g., player vs. admin).
- **Transport Layer Security (TLS):**
  All external communication secured via HTTPS or WSS with certificates managed by Let's Encrypt or a cloud ingress controller.
- **Automated Vulnerability Scanning:**
  Integration of dependency scans (via Snyk or GitHub Advanced Security) in the CI/CD pipeline.
- **Security Alerts & Monitoring:**
  Prometheus alert rules for unusual error rates, traffic anomalies, or latency spikes.
- **Regular Security Audits:**
  Periodic reviews of dependencies, Dockerfiles, and exposed network interfaces.

4. Security Architecture Diagram


Client Zone
Browser
HTTPS / WSS
Public Zone
Gateway
Input validation
Session handling
Rate limiting
Publish/Subscribe events
Internal Services Zone
NATS
Internal Pub/Sub topics
game.start
game.result
Lobby Service
Matchmaking
Lobby management
Game Service
Authoritative logic
Move validation
Data & Monitoring Zone
PostgreSQL Database
Persistent data
Restricted access
Prometheus
Metrics collection
Grafana
Dashboards