Threat	Implementation in Our System (How We Provided)
Spoofing	We implemented JWT-based Authentication to verify user identities. Passwords are never stored in plain text; we utilize BCrypt/Argon2 hashing to protect user credentials.
Tampering	Data integrity is ensured by Entity Framework Core's transactional integrity. Furthermore, the database is isolated within a Docker Volume, preventing direct external access to the fitnessreservation.db file.
Repudiation	For every successful reservation, our backend generates a unique Audit Log entry and a Transaction ID. This provides non-repudiation, ensuring that every booking action is traceable to a specific user.
Information Disclosure	We strictly use Data Transfer Objects (DTOs). This prevents sensitive database schemas or internal IDs from being exposed to the client in API responses.
Denial of Service	We validated our system's limits through k6 stress testing. Our Chaos Engineering (Pumba) experiments proved that the system can recover automatically (Self-Healing) from infrastructure-level DoS events within seconds.
Elevation of Privilege	We implemented Role-Based Access Control (RBAC) via Middleware. A "Standard" member's token cannot access "Gold" session rates or administrative endpoints, as verified in our Postman/Newman integration tests.