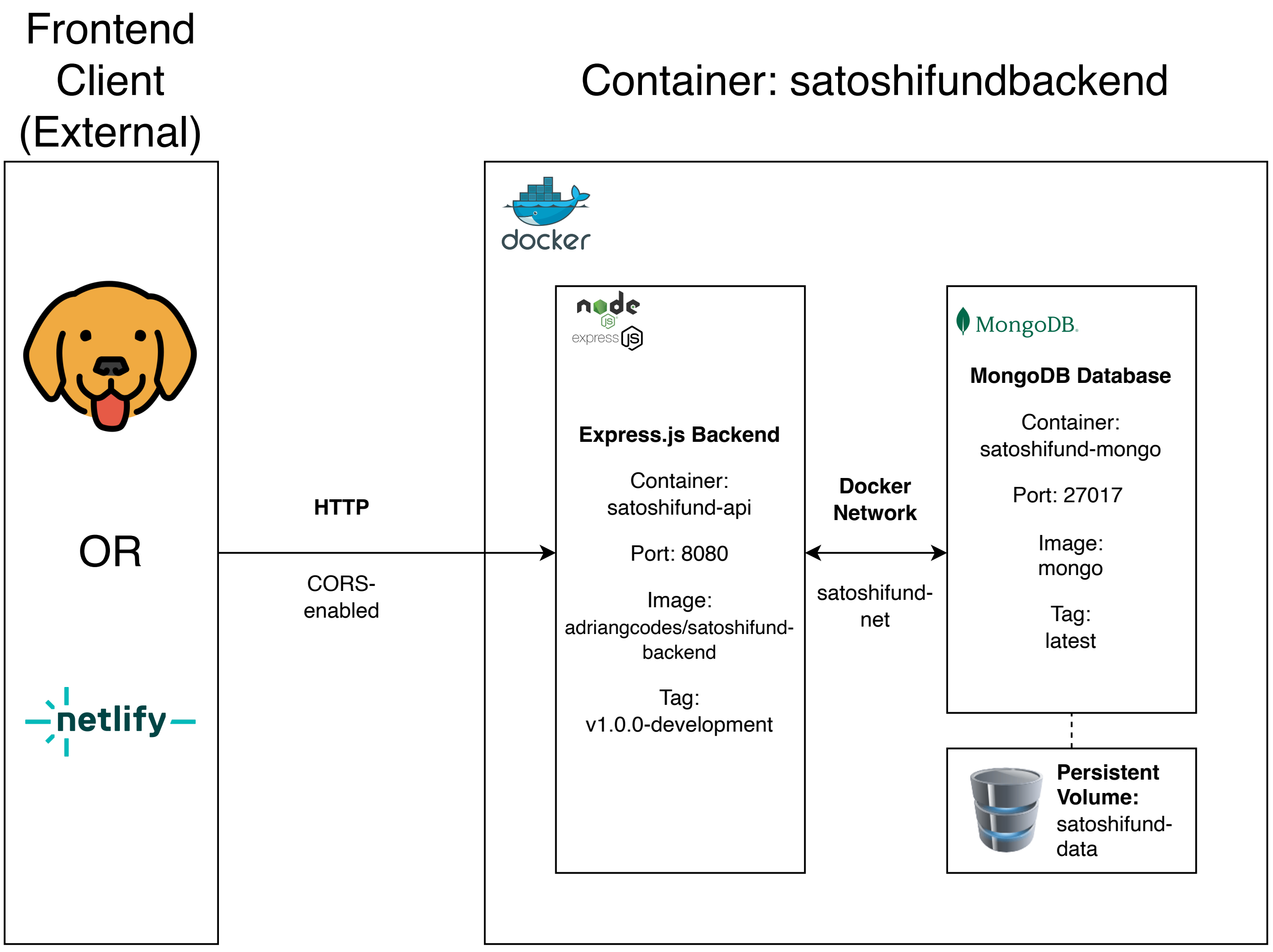
Application Architecture Diagram Overview

This containerised architecture represents a full-stack web application where a Node.js (Express) backend communicates with a MongoDB database. It is deployed using Docker Compose, allowing for easy orchestration and environment isolation.



Frontend Client

Purpose: Sends HTTP requests to the backend API.

Example: React frontend hosted on Netlify (https://satoshifund.netlify.app) or request from Bruno as per API documentation in README.

Connection: Connects to satoshifund-api via CORS (configured in the backend).

Backend API

Technology: Node.js with Express

Containerised: Via Docker using a production-ready Dockerfile.

Responsibilities:

- Handles RESTful routes for loans, users, wallets, etc.
- Authenticates users via JWT (JWT_SECRET).
- Reads environment variables securely from .env.
- Connects to MongoDB using MONGODB_URI.

Port: Exposes port 8080 to the

MongoDB

Purpose: Stores all application data including users, loans, crypto transactions, etc.

Port: Exposes 27017 for container communication only.

Docker Network: Enables seamless internal communication between API and MongoDB services. Defined under networks: in the Compose file.

Persistent Storage: Backed by satoshifund-data Docker volume, ensures persistent storage

	host machine.	across container restarts. Defined under volumes: in the Compose file.

Component Interactions

1. The frontend makes HTTP requests to the Express API, hosted on port 8080.
2. The backend verifies JWT tokens and routes requests to relevant service logic.
3. For data persistence, the backend interacts with MongoDB using the MONGODB_URI env variable.
4. Docker Compose wires the containers via a shared bridge network, and MongoDB uses a named volume for data persistence.
5. Environment-specific variables are managed via the .env file and Compose’s environment block, allowing for flexible dev/prod builds.