LastBite

Distributed food rescue system

Distributed Systems course
Alma Mater Studiorum - University of Bologna
A.A. 2024-2025

Pier Costante Babini
Marco Morelli

Concept

LastBite is a distributed full-stack web system, composed of multiple services for managing, publishing and purchasing food products remaining at the end of the day from commercial activities. The system consists primarily of the following services:

- **Backend**: Node.js/Express server, exposed as a REST API + WebSocket
- **Frontend**: Vue 3 framework
- **Database**: MongoDB for data persistence configured as a replica set
- **Reverse proxy**: Nginx for request routing and load balancing
- **Redis**: used for the distributed locking and event propagation among backend replicas
- **External API integrations**: Groq and OpenStreetMap

Use cases

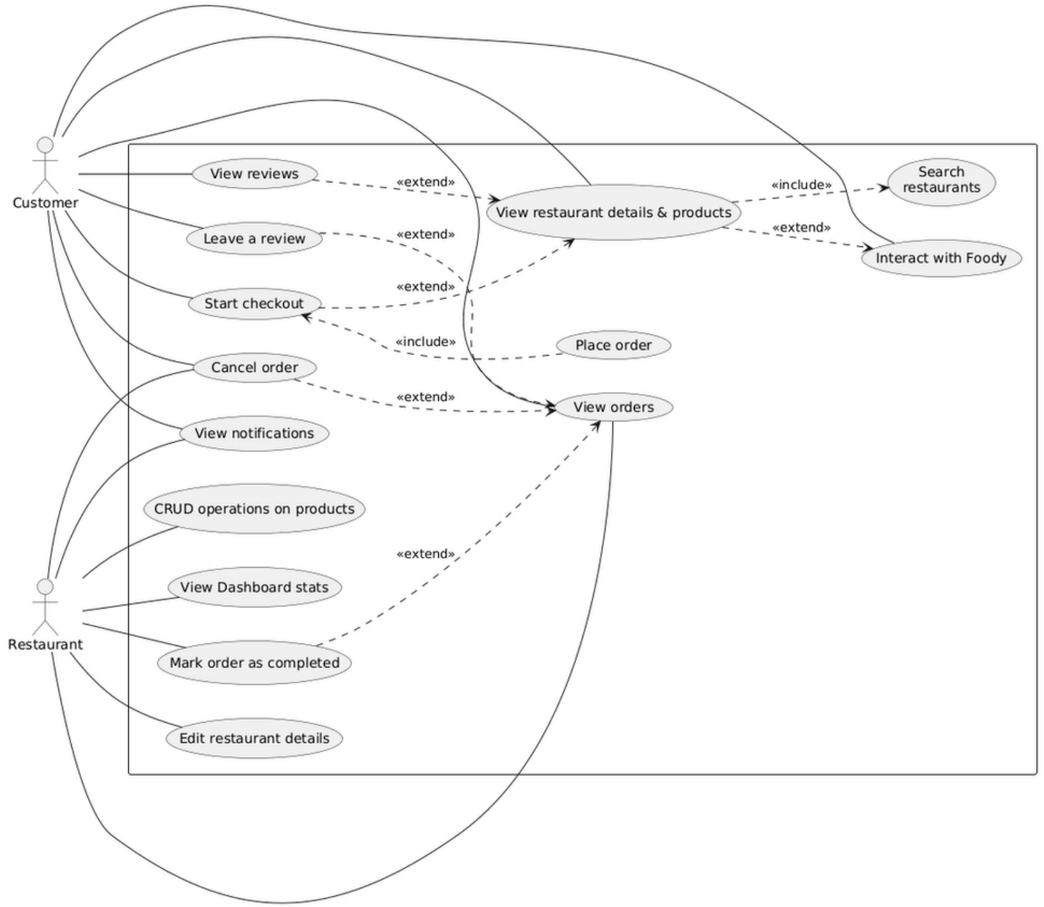


Figure 1: LastBite Actor diagram.

Functional Requirements

Customer

- Restaurant search by name
- View restaurant details
- Correct handling of products purchases
- View and cancel orders
- Review orders
- Possibility to interact with a chatbot to obtain information

Restaurant

- View restaurant statistics
- Edit restaurant info
- CRUD operations on restaurant products
- View and manage received orders

Some requirements that are common to both roles and less important (such as login and editing account info) have been omitted for simplicity.

Infrastracture

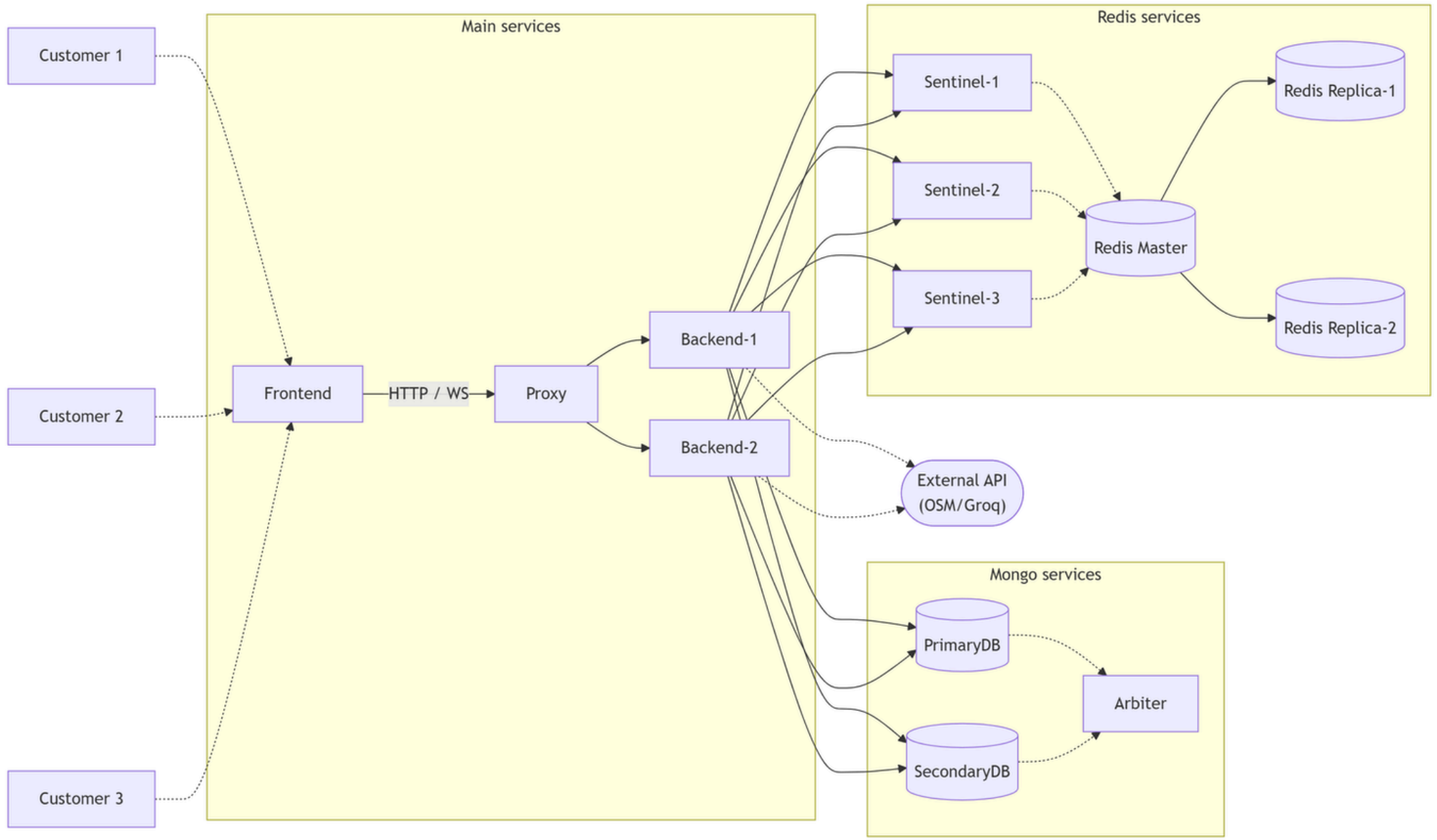


Figure 2: LastBite Infrastructure diagram.

Modelling

- **User types**: customers saved as users, restaurants saved as users with different role and an associated restaurant.
- **Ordering**: each order contains only one product.
- **Checkout handling**: the checkout process for a product is managed using Redis locks (Heartbeat + TTL).
- **Messaging**: commands and queries are handled as HTTP REST APIs, while events are handled as WebSocket events.

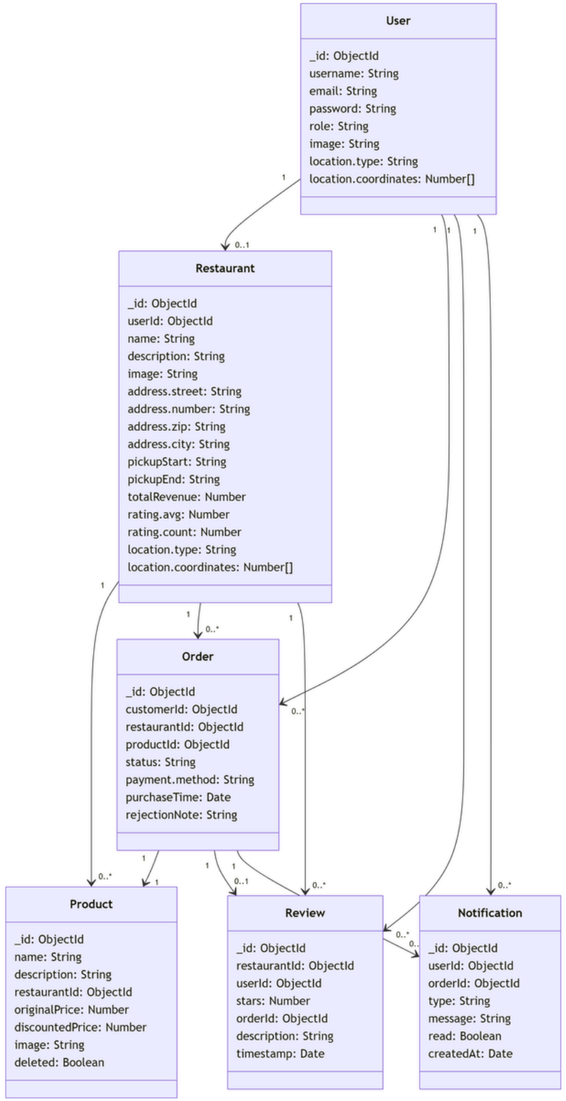


Figure 3: LastBite domain entities Class diagram.

Interaction

- **Interaction among all components**: the diagram shows the process of purchasing a product, an example of an operation that involves all the main components of the infrastructure.
- **Real-time event propagation**: the products available/in checkout are updated for all users.
- **Redis contribution**: distributed lock management, propagation of WebSocket events across multiple backends.

The example is simplified starting from a restaurant page, omitting the login phase, some failure cases and the WebSocket connection.

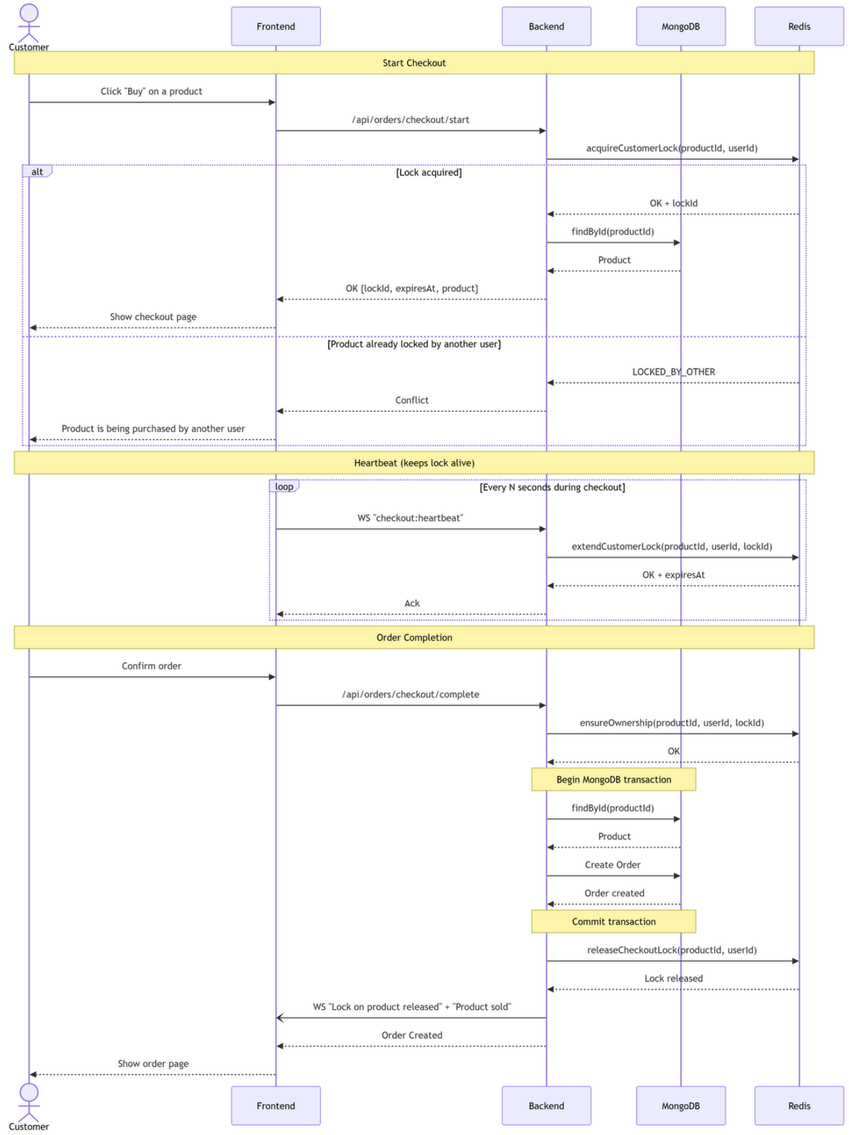


Figure 4: Sequence diagram of a product purchase.

Behaviour

- **State updates**: triggered by customer/restaurant actions in the UI.
- **Stateless**:
  - Frontend: it maintains only temporary local state, such as the authenticated user, notifications, etc.
  - Backend: contains the core logic, each request is authenticated through a JWT.
- **Stateful**:
  - MongoDB: changes to the state are triggered by the backend and saved permanently here.
  - Redis: used for temporary data, such as checkout locks with TTL, and to synchronize real-time events.

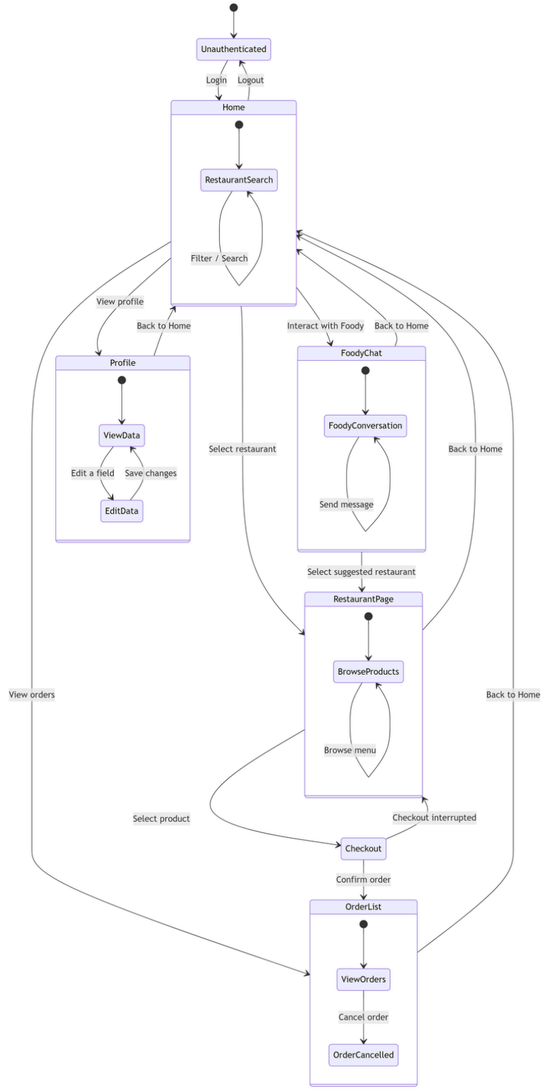


Figure 5: State diagram for the Customer user.

Preview

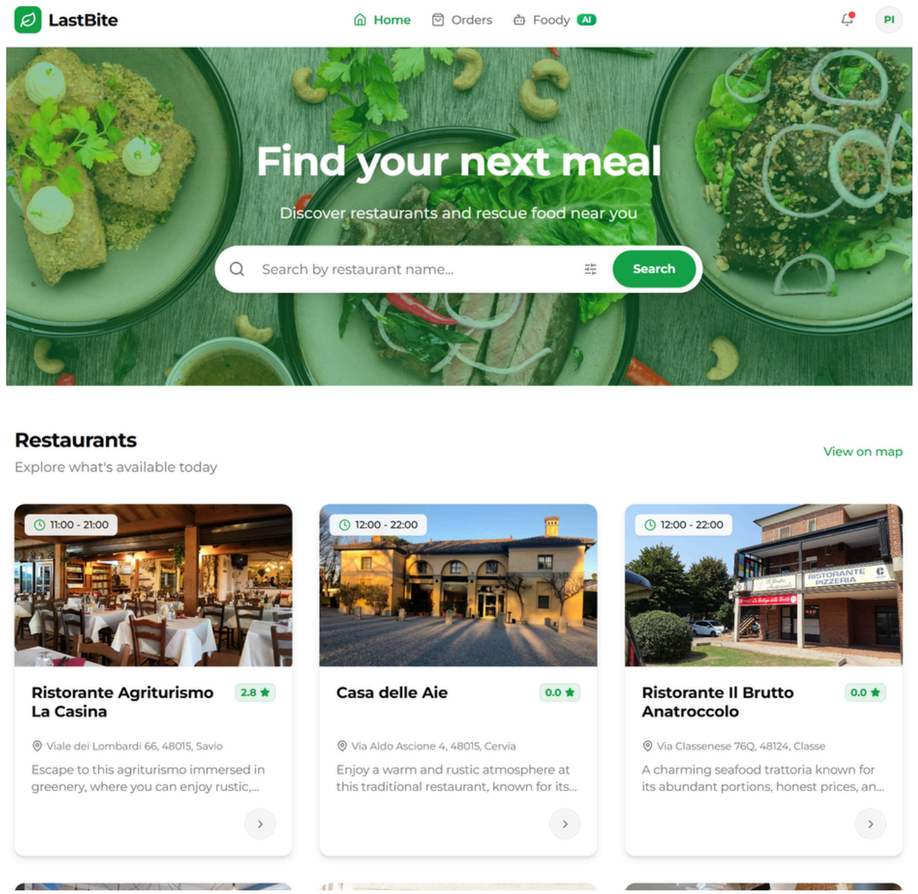
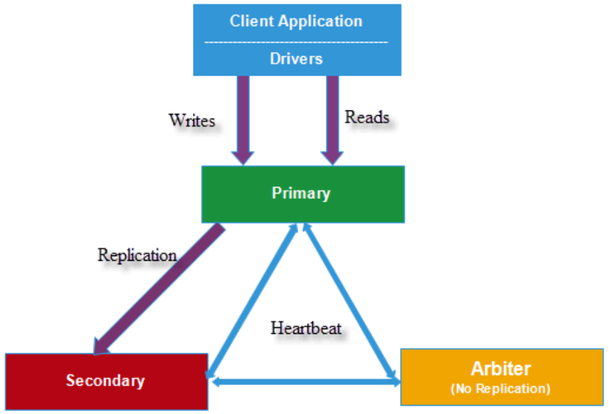


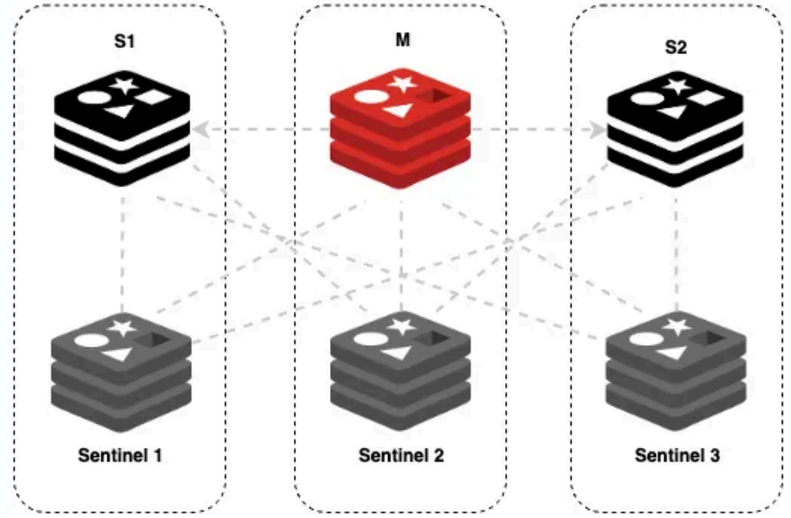
Figure 6: Customer Home page.

Fault-Tolerance

- **Database replication**: ensures secure backups and high availability. Primary DB takes precedence, replicas sync all writes/updates.
- **On primary db failure**: replicas and arbiter elect a new primary automatically.



- **Redis failover**: when the master node becomes unavailable, Sentinel promotes one of the replicas to master and the replicas synchronize with the new master.

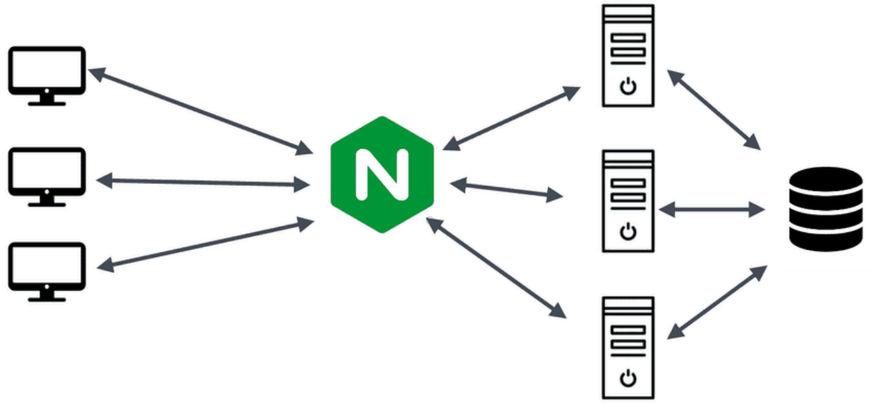


Security

- **Authentication**: required for any app access (for both HTTP and WebSocket requests).
- **Token-based auth**: JWT Token with 24h expiry, verified on each request.
- **Authorization (RBAC)**: different permissions based on the current role.
- **Passwords hashed before storage**: no plain-text credentials.
- **Network isolation**: only the frontend and proxy are exposed to the external network.

Load Balancing

- Implemented using an **NGINX proxy server**.
- **Distributes incoming requests** across two backend instances using round-robin.
- Enables **scalability** and **fault tolerance**.
- Ensures **efficient use of resources** and reduces server overload.
- Transparently **handles API routing** from the frontend to the backend.



Key technologies

- **MEVN**: full-stack app built with MongoDB, Express.js, Vue.js, Node.js for seamless and modern development.
- **Express Middlewares**: handle validation (Zod & Joi), authentication (e.g., JWT), and authorization.
- **Redis**: used for high-concurrency operations with distributed locks to avoid race conditions (e.g., product checkout).
- **Supertest + Vitest**: ensures code reliability through automated testing.
- **Socket.IO**: enables real-time bidirectional communication between client and server for features like live notifications, chat, and instant updates.

redis

socket.io

External integrations

- **Groq**: provides ultra-fast AI inference, enabling low-latency responses and scalable real-time interactions. Used for intent classification in the “Foody” chatbot.

- **OpenStreetMap**: delivers open-source geospatial data and mapping services for location-based features and route visualization. Used to convert restaurant addresses into geographic coordinates.

Testing

- Tests built with **Supertest** and **Vitest**, covering all domain entities.
- Organized in **separate test files** (e.g., user.test.js, order.test.js).
- Tests use a **simulated instance of the database**, which is cleaned after each test.
- Certain responses that depend on external integrations (such as geocoding with OSM) are **mocked** inside tests.

VITEST

Deployment

- Start by launching the **Docker daemon** (via shell or GUI).
- **Clone the repository** and set the required **environment variables** using the provided sample **env file** and the **Groq API key** if you want to enable the chatbot feature.
- Run the following **commands**:
  docker compose build
  docker compose up -d
- The app is now available at the following **address**: http://localhost:5173

Thank you for your attention