**Documentation**

**Secured REST API**

A diagram of a computer

Description automatically generated

Users interact with the system via a web browser, sending HTTP requests to a web server. This server, powered by Node.js and Express, acts as the application logic layer, handling authentication through BasicAuth middleware and managing tasks through a dedicated tasks controller. Tasks are stored in a MongoDB database, which the server accesses for data retrieval and storage operations. The server communicates with the database to perform tasks such as retrieving existing tasks, adding new tasks, and updating task completion statuses. Ultimately, the system provides users with functionality to view, add, and update tasks through a web interface, facilitating efficient task management.

**Scalable web-sockets live chat app**

We have 3 clients, a reverse proxy (in our case is HAProxy), which listens on a specified port and manages the communication between the clients and backend servers (Server 1, Server 2, and Server 3). We want the clients to communicate with each other, so the reverse proxy facilitates load balancing and directs incoming connections from clients to the appropriate backend servers. The role of Redis in this setup is as a publish-subscribe system. When Client 1 publishes a message on a specific channel, Redis ensures that clients subscribed to that channel, such as Client 2 and Client 3, immediately receive the published message. Redis handles the distribution of messages among connected clients based on their subscriptions.

A diagram of a network

Description automatically generated

**Event streaming**

**A diagram of a diagram

Description automatically generated**The provided system architecture consists of several interconnected components: a Node.js application serving as the frontend, Kafka for message streaming, and Zookeeper for managing Kafka. Users interact with the system through the Node.js application, sending HTTP requests to produce messages. The Node.js application then processes these requests and produces messages to Kafka topics. Kafka brokers manage these topics, ensuring reliable message delivery and distribution. Zookeeper serves as the coordination service for Kafka, managing broker configuration, leader election, and synchronization. This architecture enables scalable and fault-tolerant message processing, with the Node.js application handling user interactions, Kafka managing message streaming, and Zookeeper providing coordination and management capabilities for Kafka.

**Function as a Service**

**A diagram of a fire base function

Description automatically generated**

The system architecture is built around Firebase, utilizing Firebase Functions and Firestore for handling reservations and dog check-ins. Users interact with the system through HTTP requests, which are handled by Firebase Functions. These functions perform validation and processing of the requests, such as adding reservations or checking in dogs, and interact with the Firestore database to store and retrieve data. Firestore serves as the database backend, storing reservations and their associated details. This architecture enables a scalable and serverless approach to managing reservations and dog check-ins, with Firebase handling the infrastructure and scalability concerns.