# **Case Study (Work-in-Progress)**

## **Building a Scalable, Real-Time Voting Platform Using Redis 8**

### **Background**

In a world driven by interactivity and instant feedback, **voting systems are no longer limited to political elections**. From reality TV shows to product feedback campaigns, from live event polling to employee surveys — platforms today demand:

* Real-time vote aggregation
* Concurrent user handling (hundreds of thousands to millions)
* Live analytics dashboards
* Fraud prevention mechanisms
* Flexibility to scale up/down instantly

Traditional relational databases struggle under this pressure, often leading to poor user experiences, slow updates, and even system crashes. We sought a better solution.

### **The Opportunity: Redis 8 — *Beyond the Cache***

Redis is widely known as a caching system, but Redis 8 unlocks the full power of:

* **Streams** for real-time logs and data ingestion
* **Sorted Sets** for ranking vote options instantly
* **Pub/Sub** for pushing updates to frontend/UI in real-time
* **Access Control Lists (ACL)** for secure, role-based user handling
* **RedisAI (Optional)** for integrating smart models to flag abnormal patterns

This makes Redis not just fast—but **functionally deep** enough to power a **primary database and logic system** for voting.

### **The Mission**

To build a **real-time voting platform** that is:

* 🔧 **Configurable**: Any organization can set up an election/poll with custom options
* 📡 **Live**: Instant updates as users vote
* 🧠 **Smart**: AI-based fraud detection module
* ⚙️ **Tested**: Simulates real-world concurrency (up to 1 million users voting simultaneously)
* 🔐 **Secure**: Supports user authentication & authorization with rate-limiting

### **Use Cases**

This isn’t just about political elections.

|  |  |
| --- | --- |
| **Use Case** | **Redis Feature Utilized** |
| **TV show live audience votes** | Streams + Pub/Sub |
| **Corporate decision polls** | Sorted Sets + ACL |
| **Product feedback polling** | Real-time streams |
| **College elections** | Auth + Pub/Sub + Sorted Sets |
| **Employee engagement surveys** | Hash + Expiry + TTL |

The goal is to create a **plug-and-play** system that can fit any poll format with minimal customization.

### **Core System Design Principles**

* **Idempotency**: Each user can only vote once; enforced using unique keys with TTL
* **Real-Time Data Ingestion**: Redis Streams log every action with metadata
* **High Concurrency Simulation**: We use Golang to simulate 1M parallel dummy users for load testing
* **Real-Time UI Sync**: Pub/Sub system updates frontend interfaces instantly
* **Auditability**: Streams and logs can be replayed or queried for analytics
* **Pluggable AI**: RedisAI models can analyze incoming votes for anomalies, bot-like patterns, etc.

### **Simulation Focus (Upcoming)**

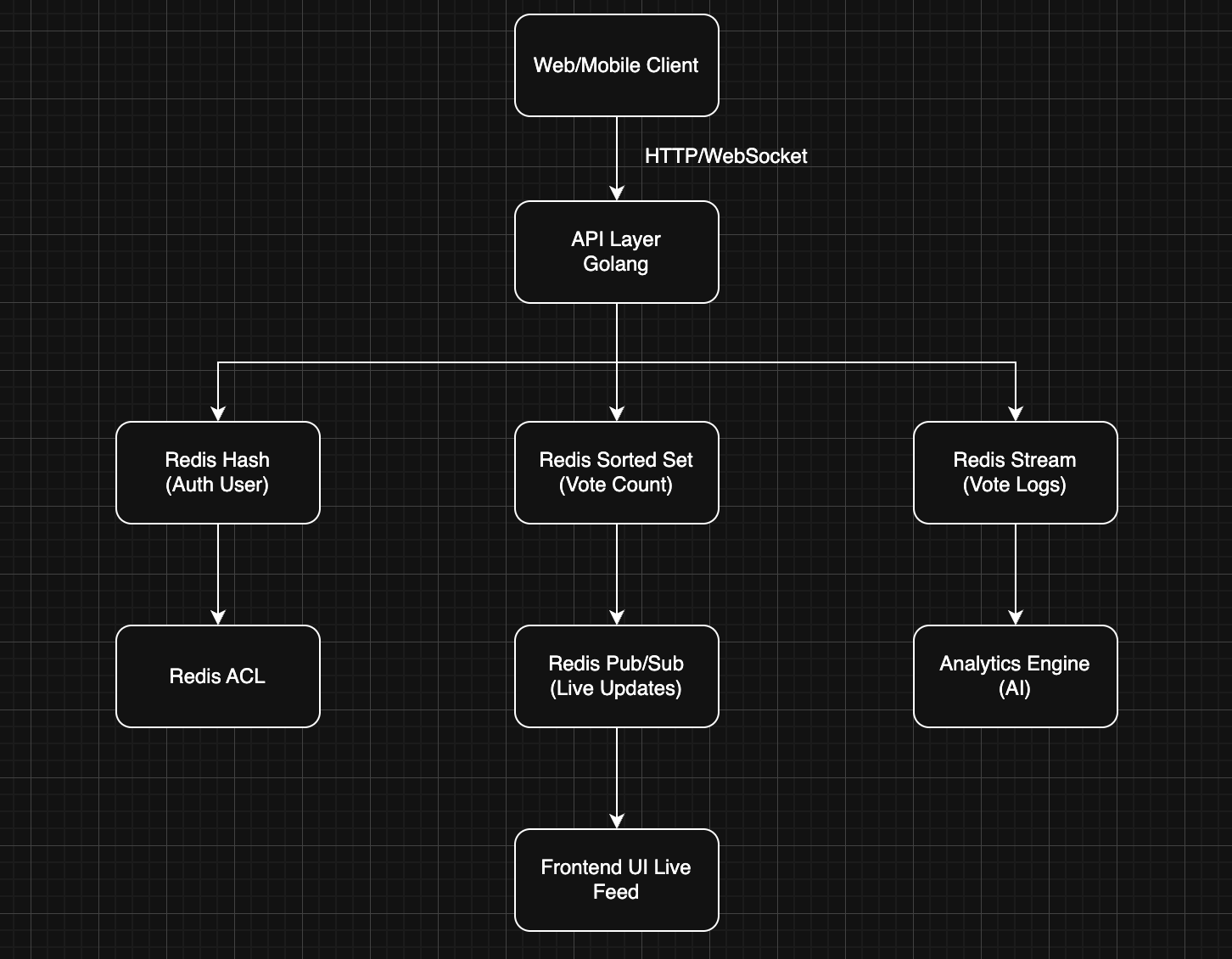
To validate Redis as a reliable backbone, we are designing stress tests:

* **Golang-based concurrent simulation** of 1 million users casting votes
* **Monitoring system performance**: latency, throughput, stream growth, Pub/Sub responsiveness
* **Validating security**: blocked repeated votes, ACL-based user permissions

### **Work-in-Progress**

We’re currently in the development and architecture phase. Our core principles are flexibility, speed, and integrity. This project aims to **redefine what real-time voting platforms can do**, powered by Redis as a primary system—not just a side cache.

**Architecture Diagram and Components**



Each Components

|  |  |
| --- | --- |
| **Component** | **Description** |
| **Web/Mobile Clients** | Users voting via browser/app |
| **API Layer** | Golang server exposing endpoints for voting, user auth, etc. |
| **Redis Hash** | Stores user credentials & profile |
| **Redis ACL** | Role-based access and permission control |
| **Redis Sorted Set** | Real-time vote counts by option |
| **Redis Streams** | Full event log of who voted and when |
| **Redis Pub/Sub** | Pushes real-time updates to UI |
| **Analytics Engine** | Optional layer that uses RedisAI or Go AI model for fraud detection |
| **Load Testing Module(Need to check how to do it)** | Golang simulation of 1M users voting in parallel |