# **Real-Time Voting System Using Go, Redis 8, and AI Enhancements**

## **Overview**

This case study explores the development of a real-time voting system built using Golang (Gin framework), Redis 8, and AI enhancements. The project is a submission under the "Beyond the Cache" category of the Redis AI Challenge 2025.

The goal is to enable fast, scalable, and intelligent polling mechanisms for high-concurrency environments like sports matches, elections, entertainment shows, live streams, and more.

## **Use Cases**

* **Live Sports Events:** Trigger polls automatically based on events like boundaries, wickets, or halftime using sentiment analysis.
* **TV Shows or Reality Contests:** Real-time voting for favorite participants.
* **Election Simulations:** Simulate real-time polling environments with massive concurrent users.
* **Conference Feedback:** Collect and visualize audience feedback instantly.
* **Classroom or Workshop Polling:** Enable educators to gather opinions quickly.

## **Architecture Structure**

real-time-voting/

├── cmd/ # Entry point

│ └── main.go # App bootstrap

├── config/ # Config loader (.env etc.)

│ └── config.go

├── internal/

│ ├── api/ # API route handlers

│ │ ├── routes.go

│ │ ├── auth.go

│ │ ├── poll.go

│ │ └── vote.go

│ ├── models/ # Struct definitions

│ │ ├── user.go

│ │ ├── poll.go

│ │ └── vote.go

│ ├── redis/ # Redis connection + queries

│ │ └── client.go

│ ├── middleware/ # JWT, Auth guards etc.

│ │ └── auth\_middleware.go

│ └── utils/ # Token generator, helpers

│ └── token.go

├── testdata/ # Load test, dummy user scripts

│ └── simulate\_votes.go

├── .env # Redis URI, JWT secret etc.

├── go.mod

└── README.md

## **Development Phases**

### **Phase 1: Project Setup**

* Scaffold the project structure.
* Integrate Redis 8 and configure connection.
* Setup Go Gin server and environment configs.

### **Phase 2: Authentication & Authorization**

* JWT-based login/signup APIs.
* Create 10 dummy users with hashed passwords.
* Middleware for protected routes.

### **Phase 3: Poll Management**

* CRUD for polls.
* Store poll metadata in Redis with TTL for active polls.
* Categorize polls: Active, Archived, Scheduled.

### **Phase 4: Voting Mechanism**

* Secure vote casting.
* Real-time Redis counters for votes.
* Prevent double voting (by user ID/IP).

### **Phase 5: High-Concurrency Simulation**

* Write scripts in testdata/simulate\_votes.go to simulate 1 million concurrent votes using goroutines.
* Benchmark Redis performance.

### **Phase 6: AI Integration (Advanced Phase)**

* **Sentiment Analysis:** Use RedisAI or Python service to analyze incoming live data (like cricket commentary).
* **Event Triggered Polls:** Based on AI analysis (e.g., sudden increase in negative sentiment after a wicket), trigger polls automatically.
* **Poll Summarization:** Use NLP to summarize results or audience reactions.

### **Phase 7: Dashboard and Visualization**

* Build a simple dashboard (React or static HTML) to display polls and results.
* Show active polls, archived stats.

### **Phase 8: Final Optimizations & Documentation**

* Improve Redis data modeling (Streams, Sorted Sets, TTLs).
* Write complete documentation.
* Dockerize the app.

## **Redis 8 Usage**

* Store poll metadata with expiry using Redis Hashes.
* Use Redis Streams to log votes.
* Store user sessions in Redis.
* Use RedisAI for live data sentiment processing.

## **AI Use Cases**

* **Real-time Sentiment Detection**: Monitor streaming input for context changes.
* **Poll Trigger Prediction**: Auto-create polls based on in-game or live event momentum.
* **Topic Clustering**: Group poll topics to avoid redundancy.
* **Bot Detection**: Use ML models to detect vote-spamming patterns.

## **Next Steps**

The project is currently in **Phase 2**, focusing on authentication and secure user onboarding. AI capabilities and real-time scale testing will be introduced in subsequent phases.

This is a living case study and will evolve through each development phase. Contributions and suggestions are welcome!