Technical Specification: Open‑Source Scrum Poker Application

## **1. Overview**

This specification details a fully open‑source technology stack and architecture for a Scrum Poker application offering:

* Real‑time updates (synchronous voting)
* Asynchronous voting windows
* AI‑driven estimation via a locally‑hosted LLM (Llama.cpp)

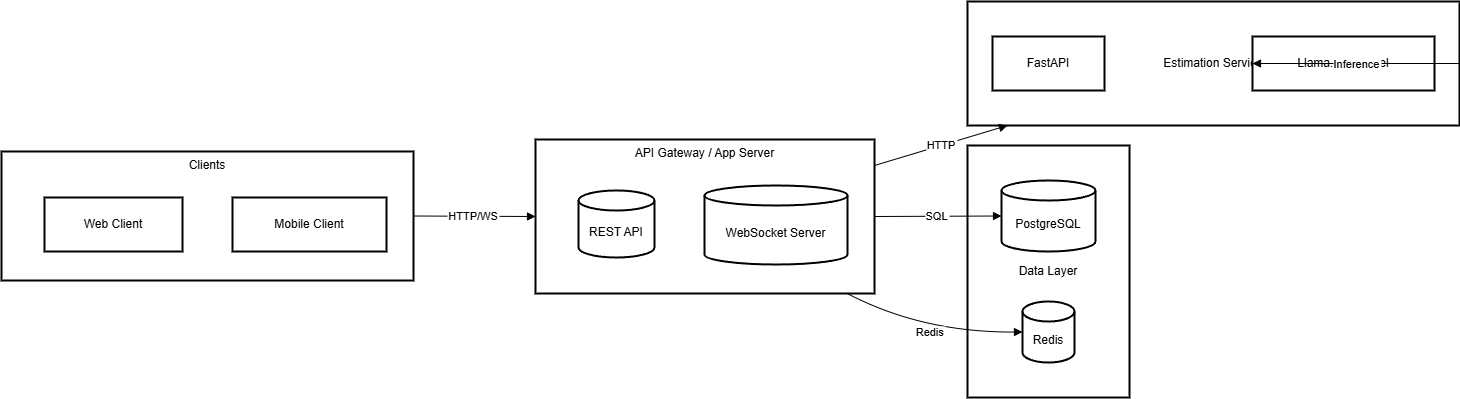
All components leverage free and open‑source software (FOSS) and libraries to ensure cost‑effectiveness, community support, and extensibility.

## **2. System Architecture**

### **2.1 High‑Level Components**

1. **Web Client (Browser)**
   * React with TypeScript
   * Vite build tool
   * Tailwind CSS for styling
   * WebSocket via Socket.IO client
2. **Mobile Client**
   * React Native (expo-managed workflow)
   * Same component logic as web where feasible
3. **API Gateway / Application Server**
   * Node.js + Express.js
   * HTTP REST endpoints
   * WebSocket server via Socket.IO
4. **AI Estimation Service**
   * Python 3.10
   * FastAPI for HTTP interface
   * Llama.cpp binding for inference of LLaMA‑family models
5. **Data Storage**
   * PostgreSQL (v15+) for persistent data
   * Redis for session caching and pub/sub
6. **Authentication & Authorization**
   * Keycloak (OpenID Connect / OAuth2) for SSO
7. **Reverse Proxy / Ingress**
   * NGINX or Traefik
8. **Containerization & Orchestration**
   * Docker Compose for local dev
   * Kubernetes (k3s or standard) for production
9. **CI/CD Pipeline**
   * GitLab CI or GitHub Actions
10. **Monitoring & Logging**
    * Prometheus + Grafana
    * Loki + Grafana

### **2.2 Logical Diagram**



## **3. Detailed Components & Libraries**

| **Component** | **Technology / Library** | **Version** | **License** | **Purpose** |
| --- | --- | --- | --- | --- |
| Frontend Framework | React | ^18.0.0 | MIT | UI components, state management |
| Build Tool | Vite | latest | MIT | Fast development server, bundling |
| Styling | Tailwind CSS | ^3.0 | MIT | Utility‑first CSS framework |
| Mobile Framework | React Native (Expo) | SDK 48 | MIT | Cross‑platform mobile app |
| HTTP Server | Express.js | ^4.18.0 | MIT | REST API |
| Real‑Time | Socket.IO | ^4.6.0 | MIT | WebSocket communication |
| AI Service Framework | FastAPI | ^0.95 | MIT | Lightweight Python API |
| LLM Inference | Llama.cpp | latest | MIT | Local inference of LLaMA models |
| Primary Database | PostgreSQL | 15.x | PostgreSQL | Persistent storage |
| Caching / Pub‑Sub | Redis | 7.x | BSD‑3 | Session caching, pub/sub for notifications |
| Authentication Server | Keycloak | 21.x | Apache‑2.0 | Identity provider, SSO, RBAC |
| Reverse Proxy | NGINX or Traefik | latest | BSD / MIT | TLS termination, routing |
| Containerization | Docker | latest | Apache‑2.0 | Packaging microservices |
| Orchestration | Kubernetes (k3s / vanilla) | 1.27+ | Apache‑2.0 | Container orchestration |
| CI/CD | GitLab CI / GitHub Actions | N/A | MIT / MIT | Automated build, test, deploy |
| Metrics & Logging | Prometheus, Grafana, Loki | latest | Apache‑2.0 | System monitoring, log aggregation |

## **4. API Endpoints**

### **4.1 REST API**

| Method | Path | Description | Request Body | Response |
| --- | --- | --- | --- | --- |
| GET | /api/sessions | List active sessions | — | Array of session summaries |
| POST | /api/sessions | Create new session | { title, mode, deadline } | Created session object |
| GET | /api/sessions/:id | Get session details | — | Session object with stories, votes |
| POST | /api/sessions/:id/vote | Submit async vote | { userId, storyId, voteValue } | Success status |
| GET | /api/sessions/:id/results | Fetch current votes & AI suggestions | — | { votes[], suggestions[] } |
| POST | /api/sessions/:id/terminate | Close session (trigger summary export) | — | Export URL / summary object |
| POST | /api/auth/token | Exchange Keycloak code for JWT | { code } | { access\_token, refresh\_token } |

### **4.2 WebSocket Events**

| Event | Payload | Description |
| --- | --- | --- |
| connect | — | Client connected |
| join-session | { sessionId, token } | Join WebSocket room |
| vote-submitted | { storyId, userId, voteValue } | Broadcast vote submission |
| reveal-votes | { sessionId } | Trigger reveal of all votes |
| session-updated | { sessionData } | Broadcast updated session state |
| disconnect | — | Client disconnected |

## **5. Data Model (PostgreSQL)**

-- Users

CREATE TABLE users (

user\_id UUID PRIMARY KEY,

username TEXT UNIQUE NOT NULL,

email TEXT UNIQUE NOT NULL,

role TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT now()

);

-- Sessions

CREATE TABLE sessions (

session\_id UUID PRIMARY KEY,

title TEXT NOT NULL,

mode TEXT CHECK(mode IN ('realtime','async')) NOT NULL,

deadline TIMESTAMP,

created\_by UUID REFERENCES users(user\_id),

created\_at TIMESTAMP DEFAULT now()

);

-- Stories

CREATE TABLE stories (

story\_id UUID PRIMARY KEY,

session\_id UUID REFERENCES sessions(session\_id),

title TEXT NOT NULL,

description TEXT,

created\_at TIMESTAMP DEFAULT now()

);

-- Votes

CREATE TABLE votes (

vote\_id UUID PRIMARY KEY,

story\_id UUID REFERENCES stories(story\_id),

user\_id UUID REFERENCES users(user\_id),

vote\_value INT NOT NULL,

submitted\_at TIMESTAMP DEFAULT now()

);

-- AI Suggestions

CREATE TABLE ai\_suggestions (

suggestion\_id UUID PRIMARY KEY,

story\_id UUID REFERENCES stories(story\_id),

suggestion\_value INT NOT NULL,

confidence REAL,

generated\_at TIMESTAMP DEFAULT now()

);

## **6. Deployment & Operations**

1. **Local Development**: Docker Compose orchestrates services: PostgreSQL, Redis, Keycloak, API server, AI service, Web client.
2. **Staging/Production**: Kubernetes deployment with Helm charts; separate namespaces for staging and prod.
3. **CI/CD**: On push to main, run lint, unit tests, build images, push to registry, deploy to staging. On tag release, deploy to production.
4. **Monitoring**: Prometheus scrapes metrics endpoints; Grafana dashboards visualize session counts, API latency, error rates. Loki captures logs for troubleshooting.
5. **Backups**: Daily PostgreSQL dumps; Redis RDB snapshots every hour; store in object storage.

## **7. Security Considerations**

* Enforce HTTPS with TLS certificates (Let’s Encrypt)
* Use Keycloak for centralized auth; short-lived JWTs
* Protect WebSocket endpoints via token verification
* Encrypt sensitive data at rest (PG PGP encryption for user emails if required)
* Role‑based access: only Scrum Masters can create/terminate sessions
* Regular vulnerability scanning (Trivy, Clair)

## **8. Testing Strategy**

* **Unit Tests**: 80%+ coverage for API and AI service
* **Integration Tests**: Simulate full vote flows (real‑time & async)
* **Performance Tests**: Load testing WebSocket server with k6 (10k concurrent clients)
* **Security Tests**: OWASP ZAP scans, JWT token tampering checks

## **9. Maintenance & Support**

* Open‑source repository on GitHub with permissive license (MIT)
* Contribution guidelines and issue templates
* Community Slack or Discord channel for support
* Versioning Semantic (MAJOR.MINOR.PATCH)