Low‑Level Technical Specification: Frontend

## **1. Project Structure**

bash

CopyEdit

/src

/assets # static images, fonts

/components # reusable presentational & container components

/Card

Card.tsx

Card.test.tsx

Card.module.css

/VoteList

/AIHint

/ErrorBoundary

/hooks # custom React hooks

useSocket.ts

useAsyncVoting.ts

useAIHint.ts

/models # TypeScript interfaces & types

Session.ts

Vote.ts

Suggestion.ts

/pages # page-level components (React Router)

HomePage.tsx

SessionPage.tsx

NotFound.tsx

/services # HTTP & WebSocket service clients

api.ts

socket.ts

auth.ts

/state # global state management (e.g. Zustand or Redux)

store.ts

sessionSlice.ts

/styles # Tailwind overrides & global CSS

tailwind.config.js

globals.css

/utils # pure helper functions

date.ts

validators.ts

index.tsx # React entry point

App.tsx # root component & routing

## **2. Dependencies & Versions**

* **Core:** • react v18.x (MIT)  
   • react-dom v18.x (MIT)  
   • react-router-dom v6.x (MIT)
* **State Management:** • zustand v4.x (MIT) *or* @reduxjs/toolkit v1.x (MIT)
* **Real‑Time:** • socket.io-client v4.x (MIT)
* **HTTP:** • axios v1.x (MIT)
* **Styling:** • tailwindcss v3.x (MIT)  
   • postcss v8.x (MIT), autoprefixer v10.x (MIT)
* **Testing:** • jest v29.x (MIT), @testing-library/react v14.x (MIT)  
   • cypress v12.x (MIT) for E2E
* **Utilities:** • dayjs v1.x (MIT) for date formatting  
   • clsx v1.x (MIT) for conditional classNames

## **3. Component Specifications**

### **3.1 Card Component**

* **Props** • value: number \| '∞'  
   • selected: boolean  
   • disabled: boolean  
   • onSelect(value: number \| '∞'): void
* **Behavior** • Renders a clickable/tappable card.  
   • Applies border-blue-500 bg-blue-50 when selected.  
   • Calls onSelect on click if not disabled.
* **Accessibility** • role="button"  
   • tabIndex=0  
   • aria-pressed={selected}  
   • Key handlers for Enter/Space.

### **3.2 VoteList Container**

* **Responsibilities** • Fetches stories & votes from global state.  
   • Renders <Card> for each voting option.  
   • Submits via Socket.IO or REST based on mode.
* **State** • Local selectedValue: number \| null.
* **Side‑Effects** • Calls socket.emit('vote-submitted', { storyId, userId, vote }).  
   • On async mode, calls api.submitVote(...).

### **3.3 AIHint Component**

* **Props** • storyId: string
* **Behavior** • On mount, fetches suggestion via api.getAISuggestion(storyId).  
   • Displays spinner until resolved.  
   • Shows suggestion value & confidence badge.

### **3.4 ErrorBoundary**

* Catches render errors in its subtree.
* Renders fallback UI with “Try again” button.
* Logs error via utils.logError(error).

## **4. State Management**

### **4.1 Store Shape (Zustand Example)**

ts

CopyEdit

interface SessionState {

session: Session | null;

votes: Record<string, Vote[]>;

suggestions: Record<string, Suggestion>;

loading: boolean;

error: string | null;

setSession(data: Session): void;

addVote(vote: Vote): void;

setSuggestion(s: Suggestion): void;

fetchSession(id: string): Promise<void>;

}

### **4.2 Actions**

* fetchSession(id) → loads session & stories via api.getSession, populates state.
* addVote → updates local votes and notifies UI.
* setSuggestion → stores AI suggestion per story.

## **5. API Integration**

### **5.1 Axios Instance (services/api.ts)**

ts

CopyEdit

import axios from 'axios';

export const api = axios.create({

baseURL: import.meta.env.VITE\_API\_URL,

timeout: 5000,

});

api.interceptors.request.use((config) => {

const token = localStorage.getItem('accessToken');

if (token) config.headers.Authorization = `Bearer ${token}`;

return config;

});

### **5.2 Key Endpoints**

* GET /sessions/:id → fetch session metadata & stories
* POST /sessions/:id/vote → async vote submission
* GET /sessions/:id/results → current votes + AI suggestions
* GET /stories/:storyId/suggestion → AI estimate for story

## **6. Real‑Time Updates**

### **6.1 Socket Module (services/socket.ts)**

ts

CopyEdit

import { io, Socket } from 'socket.io-client';

let socket: Socket;

export function connect(userToken: string) {

socket = io(import.meta.env.VITE\_WS\_URL, {

auth: { token: userToken },

reconnectionAttempts: 5,

transports: ['websocket'],

});

return socket;

}

export function joinSession(sessionId: string) {

socket.emit('join-session', { sessionId });

}

export function onVote(cb: (vote: Vote) => void) {

socket.on('vote-submitted', cb);

}

// handle other events: 'session-updated', 'reveal-votes'

* **Reconnection:** • reconnectionAttempts: 5  
   • Exponential backoff default.

## **7. Async Voting UI**

* **Deadline Timer** (hooks/useAsyncVoting.ts):  
   • Calculates remaining time until session.deadline.  
   • Triggers 12h and 1h reminders via Notification API.
* **Partial Results View:** • Badge next to each story: ✓ if voted, … if pending.

## **8. Styling & Theming**

* **Tailwind Config** (tailwind.config.js):  
   • Enable JIT mode, purge unused CSS from src/\*\*/\*.{tsx,ts}.  
   • Define custom colors (Eduvibz Green/Blue).  
   • Enable darkMode: 'class'.
* **Global CSS** • Import Tailwind base, components, utilities in globals.css.

## **9. Routing**

**React Router (v6)** in App.tsx:  
  
 tsx  
CopyEdit  
<BrowserRouter>

<Routes>

<Route path="/" element={<HomePage />} />

<Route path="/session/:id" element={<SessionPage />} />

<Route path="\*" element={<NotFound />} />

</Routes>

</BrowserRouter>

* **Guards** • Redirect to login if !auth.isAuthenticated().

## **10. Error Handling & Notifications**

* **Global ErrorBoundary** wraps <App />.
* **Toast Notifications** via a lightweight library (e.g., react-hot-toast) for API errors, vote confirmations.
* **Form Validation** using Zod or Yup schemas in pages (e.g., session creation).

## **11. Performance Optimizations**

* **Code‑Splitting** • React.lazy & Suspense for SessionPage and large components.
* **Memoization** • Use React.memo on pure components.  
   • useCallback for event handlers.
* **Virtualization** • Use react-window if story list > 50.
* **Image Optimization** • Serve SVG icons or compressed PNGs.

## **12. Accessibility (WCAG 2.1 AA)**

* **Keyboard Navigation:** • All interactive elements (<Card>, buttons) focusable & operable via keyboard.
* **ARIA Labels:** • Provide aria-label for emoji or icon-only buttons.
* **Color Contrast:** • Ensure text ≥4.5:1 contrast ratio.
* **Screen‑Reader Support:** • Use semantic HTML; test with NVDA or VoiceOver.

## **13. Offline & Resilience**

* **Service Worker (Optional PWA)** • Cache core assets & app shell.  
   • Fallback to cached session state if offline for <5 minutes.
* **Local Storage** • Temporarily store pending async votes when offline, auto-sync on reconnect.

## **14. Testing Strategy**

* **Unit Tests** • Jest + RTL: test components, hooks (useSocket, useAsyncVoting).
* **Integration Tests** • Mount VoteList + mocked Socket.IO server (using socket.io-mock).
* **E2E Tests** • Cypress: simulate live voting flow, async window, AI hint display.
* **Performance** • Lighthouse CI GitHub Action to track bundle size & performance metrics.

## **15. Build & Deployment**

* **Vite Configuration** • define: { 'process.env': {} } for compatibility.  
   • build.rollupOptions: split vendor code.
* **Environment Variables** • VITE\_API\_URL, VITE\_WS\_URL, VITE\_KEYCLOAK\_URL.
* **Deploy** • Serve static build via NGINX or a CDN (e.g., GitHub Pages for dev).  
   • Use Docker multistage build to produce optimized container image.