AQUA

Technical Architecture & Implementation

AI-Powered Call Center Audit System

Hackathon Technical Deep Dive

Agenda

1. **Requirements Overview** - What we needed to build
2. **Technology Stack** - Why we chose each technology
3. **System Architecture** - Four-tier design
4. **Implementation Highlights** - Key features built
5. **AI-Assisted Development** - Three-agent workflow
6. **Quality Metrics** - Tests, coverage, code quality
7. **Demo** - Live system walkthrough

Requirements Overview

Hackathon Challenge

Category	Requirements
Core Features	Call library, detail view, audio player, transcripts
Analysis	AI scoring, sentiment analysis, anomaly detection
Admin	RBAC, teams, companies, projects, roles management
Constraints	Desktop-only, English-only, REST APIs, no WebSockets
Timeline	2 days

Goal: Build a complete frontend for AI-powered call center auditing

Requirements Prioritization

MoSCoW Method Applied

Priority	Features	Count
P0 - Must Have	Call library, detail, audio player, RBAC, login	8
P1 - Should Have	Upload, analytics, notifications, filters	4
P2 - Could Have	Dashboard, team/company/role management	5
TOTAL		**17**

Result: 100% completion (17/17 features delivered)

Technology Stack Decision

Framework Selection: React + Vite

Criteria	React + Vite	Next.js	Vue
Setup Time	**2 min**	5 min	3 min
HMR Speed	**Instant**	Fast	Fast
SSR Needed?	No ✓	Overkill	No
Audio Libraries	**Best**	Same	Limited
UI Library	**shadcn/ui**	Same	Different

Decision: React 19 + Vite 7 (fastest dev experience, best ecosystem)

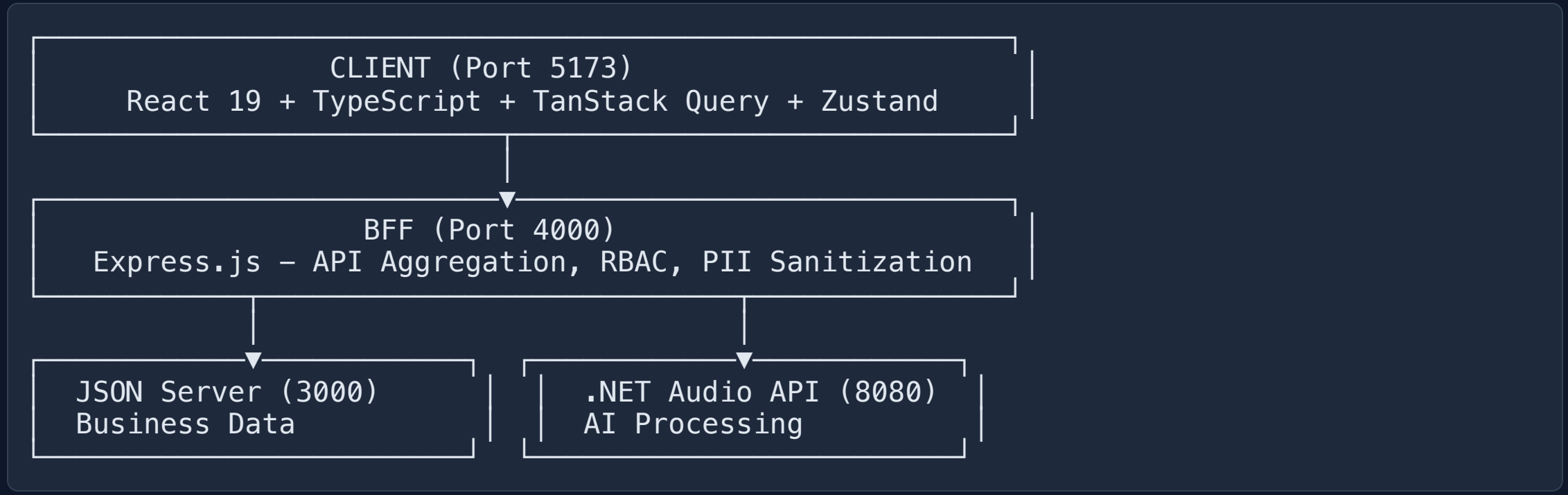
Complete Technology Stack

Frontend Technologies

Layer	Technology	Version	Purpose
Framework	React	19.2.0	UI rendering
Language	TypeScript	5.9.3	Type safety
Build	Vite	7.2.4	Fast bundling
Styling	Tailwind CSS	4.1.10	Utility-first CSS
Components	shadcn/ui + Radix	Latest	Accessible UI
Server State	TanStack Query	5.80.7	API caching
Client State	Zustand	5.0.9	Simple state
Routing	React Router	7.6.1	Navigation
Audio	WaveSurfer.js	7.x	Waveform
Charts	Recharts	3.5.1	Analytics

Architecture Overview

Four-Tier System Design



Why BFF Layer?

Backend For Frontend Benefits

Problem	BFF Solution
Multiple APIs	Single entry point
CORS issues	Centralized config
Security	RBAC middleware
PII exposure	Data sanitization
Notifications	Long-polling service

📁 **aqua-bff/src/index.ts**

```
// API Aggregation Example
app.use('/api', proxy({ target: 'http://localhost:3000' }))
app.use('/audio-api', proxy({ target: 'http://localhost:8080' }))
```

State Management Strategy

Server vs Client State Separation

TanStack Query (Server State)	Zustand (Client State)
• API data (calls, users, companies) • Caching (5min stale) • Loading/error states • Background refetch • Optimistic updates	• UI state (sidebar, filters, modals) • Auth (user, role, permissions) • Audio playback state • Theme preference

Why? Clear separation = easier debugging & testing

Project Structure

Feature-First Architecture

📁 src/

```
src/
├── components/          # Shared UI (70+ components)
│   ├── ui/              # shadcn/ui (25+ base components)
│   ├── layout/          # Sidebar, Header, MainLayout
│   └── audio/           # AudioPlayer, Waveform
├── features/            # Self-contained modules
│   ├── calls/           # CallsPage + CallsTable
│   ├── call-detail/     # Detail + Summary/Transcript/Overrides
│   ├── analytics/       # Dashboard + Charts
│   └── [admin]/         # teams, companies, projects, roles
├── services/api/        # Axios clients + endpoints
├── stores/              # Zustand (auth, app, audio, theme)
├── hooks/               # Custom hooks (useCalls, useDebounce)
└── types/               # TypeScript interfaces
```

Key Implementation: Audio Player

WaveSurfer.js Integration

📁 **src/components/audio/SentimentAudioPlayer.tsx**

```
// Sentiment-colored waveform regions
const sentimentColors = {
  Positive: 'rgba(34, 197, 94, 0.4)',   // Green
  Neutral: 'rgba(234, 179, 8, 0.4)',    // Yellow
  Negative: 'rgba(239, 68, 68, 0.4)',   // Red
}

// Create regions from diarization data
diarization.forEach(turn => {
  wavesurfer.addRegion({
    start: turn.startTime,
    end: turn.endTime,
    color: sentimentColors[turn.sentiment],
  })
})
```

Key Implementation: RBAC

Role-Based Access Control

📁 **src/stores/auth-store.ts**

```
// 6 Roles with Permission Sets
const ROLE_PERMISSIONS = {
  'Entity Administrator': ['users', 'scorecard'],
  'Super Admin': ['teams', 'companies', 'projects', 'roles'],
  'Client Stakeholder': ['monitor', 'reports'],
  'Support Supervisor': ['monitor', 'reports', 'exportinfo'],
  'Support Agent': ['reviewcalls', 'coachingcalls'],
  'QC Analyst': ['upload', 'reviewcalls', 'score', 'notes'],
}

// Permission check hook
const hasPermission = (permission: string) =>
  userPermissions.includes(permission)
```

Key Implementation: Data Fetching

TanStack Query Patterns

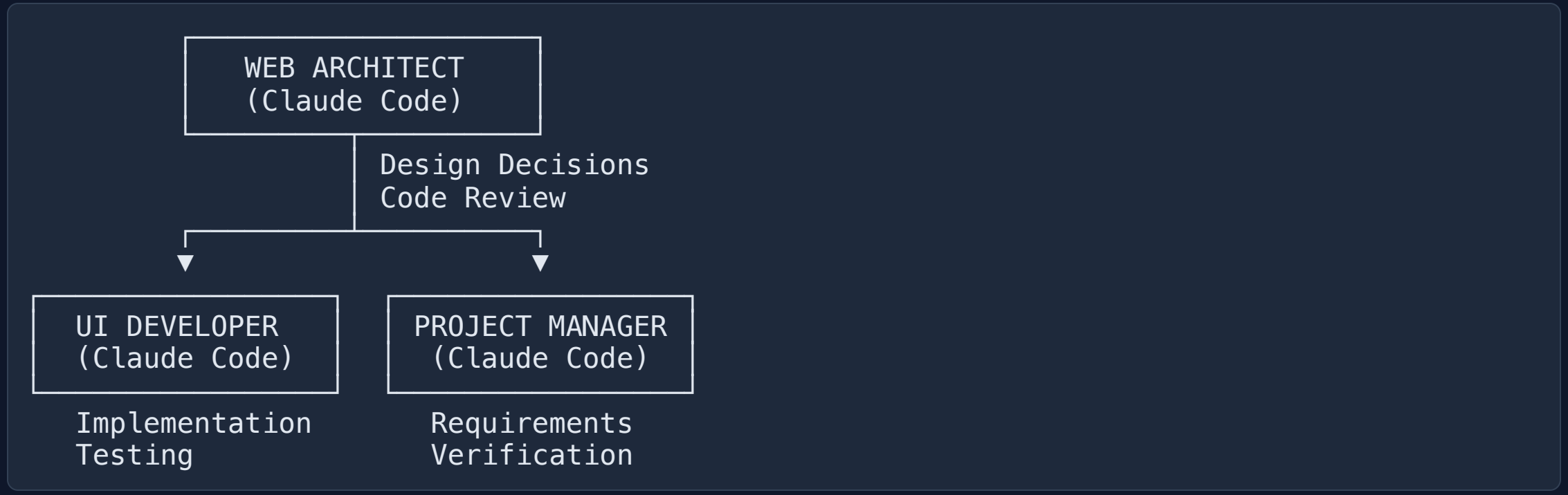
📁 **src/hooks/use-calls.ts**

```
// Query Key Factory Pattern
export const callsKeys = {
  all: ['calls'] as const,
  lists: () => [...callsKeys.all, 'list'] as const,
  list: (filters?: Filters) => [...callsKeys.lists(), filters],
  detail: (id: string) => [...callsKeys.all, 'detail', id],
}

// Custom Hook with Caching
export const useCallsQuery = (filters?: Filters) => {
  return useQuery({
    queryKey: callsKeys.list(filters),
    queryFn: () => callsApi.getSummary(filters),
    staleTime: 5 * 60 * 1000, // 5 minutes
  })
}
```

AI-Assisted Development

Three-Agent Collaboration Model



Agent Responsibilities

Division of Work

Agent	Responsibilities	Deliverables
Web Architect	Framework, architecture, code review	Docs, decisions
UI Developer	Components, features, testing	70+ components, 87 tests
Project Manager	Requirements, gap analysis	Checklists, reports

Benefit: Parallel work streams, consistent patterns, comprehensive coverage

AI Development Impact

Speed & Quality Metrics

Task	Traditional	AI-Assisted	Improvement
Project Setup	4 hours	30 min	**8x faster**
Component Scaffolding	2 hr/comp	15 min	**8x faster**
Documentation	8 hours	2 hours	**4x faster**
Bug Investigation	Variable	Minutes	**Significant**

Total: 15,000+ lines of production code in 48 hours

Testing Strategy

Test Distribution

Test File	Tests	Category
auth-store.test.ts	16	State Management
SentimentAudioPlayer.test.tsx	19	Audio Components
CallsTable.test.tsx	16	Feature Components
use-calls.test.tsx	11	Custom Hooks
calls.api.test.ts	6	API Integration
Others	19	Various
TOTAL	**87**	

Tools: Vitest + React Testing Library + MSW

Code Quality Metrics

Build & Performance

Metric	Value
TypeScript Strict Mode	Enabled
Lines of Code	15,000+
Components	70+
Tests Passing	87
Build Time	< 3 seconds
Bundle Size (gzip)	~150 KB JS
Dev Server Start	< 1 second

API Coverage

Endpoints Implemented

Endpoint	Method	Used In
`/Calls`	GET	CallDetailPage
`/CallSummary`	GET	CallsPage, Dashboard
`/Profiles`	GET	LoginPage
`/Companies`	CRUD	CompaniesPage
`/Projects`	CRUD	ProjectsPage
`/Teams`	CRUD	TeamsPage
`/Roles`	GET	RolesPage
`/Agents`	CRUD	Team management
`/IngestAudio`	POST	Upload modal

Coverage: 12/12 endpoints (100%)

Technical Constraints Compliance

All Requirements Met

Constraint	Status	Implementation
Desktop-only	✅	Fixed layouts, no mobile
English-only	✅	No i18n libraries
REST APIs only	✅	Axios, no GraphQL
Long-polling	✅	BFF notification service
No WebSockets	✅	Polling in notifications.api
RBAC	✅	6 roles, permission gates
No PII exposure	✅	BFF sanitization layer

Feature Completion Summary

All User Stories Delivered

FEATURE COMPLETION MATRIX			
Priority	Total	Implemented	Status
P0 P1 P2	8 4 5	8 4 5	✅ Complete ✅ Complete ✅ Complete
TOTAL	17	17	100% Delivered

Key Technical Decisions

What Made This Possible

1. **React + Vite** - Instant HMR, fast iteration
2. **TanStack Query** - Smart caching reduced API complexity
3. **Zustand** - Minimal boilerplate for client state
4. **shadcn/ui** - Copy-paste components, full control
5. **Feature-first structure** - Clear ownership, easy navigation
6. **BFF layer** - Clean API aggregation, security
7. **AI collaboration** - 3-agent workflow for parallel work

Lessons Learned

What Worked Well

- **AI-first approach** - Claude Code accelerated everything
- **Architecture upfront** - Solid foundation prevented rewrites
- **Feature modules** - Easy to parallelize work
- **Type safety** - TypeScript caught bugs early

Challenges Overcome

- Audio player cross-browser compatibility
- Complex state sync (audio + transcript)
- Dark mode theming consistency

Live System URLs

Running Services

Service	Port	Purpose
Frontend	5173	React SPA
BFF	4000	API Gateway
JSON Server	3000	Mock Data
Audio API	8080	AI Processing

📁 start-servers.sh

```
# Start all services
./start-servers.sh
```

Questions?

Technical Deep Dive Available

Documentation:

- `/project-documentation/architecture-documentation.md`
- `/project-documentation/architecture-decisions.md`
- `/project-documentation/requirements-verification-report.md`

Key Metrics:

- 15,000+ lines TypeScript
- 87 tests passing
- 100% feature completion

Appendix: TypeScript Interfaces

Core Data Models

📁 **src/types/call.ts**

```typescript
interface Call {
  transactionId: string
  callId: string
  agentName: string
  transcription: Transcription
  scoreCard: ScoreCard
  sentimentAnalisys: SentimentAnalysis
  anomaly: Anomaly
}

interface Anomaly {
  flag: 'Red' | 'Yellow' | 'Green'
  justification: string[]
}
```

Appendix: Component Examples

Audio Player Props

📁 **src/components/audio/SentimentAudioPlayer.tsx**

```
interface SentimentAudioPlayerProps {
  audioUrl: string
  diarization: DiarizationEntry[]
  onTimeUpdate?: (time: number) => void
  onPlayStateChange?: (playing: boolean) => void
}

interface DiarizationEntry {
  turnIndex: number
  speaker: 'Agent' | 'Customer'
  text: string
  sentiment: 'Positive' | 'Neutral' | 'Negative'
  startTime: number
  endTime: number
}
```