Diego Portfolio - Product Requirements Document (PRD)

Goals and Background Context

Goals

- Create a unique, memorable portfolio website that showcases technical skills through an innovative music-themed experience
- Demonstrate full-stack capabilities (React, Python, AI/ML, FastAPI, LangChain) through the portfolio itself
- Tell Diego's professional story in a cohesive, engaging way that stands out from traditional portfolios
- Provide recruiters with easy access to project evidence, skills, and contact information
- Create an immersive experience that blends technology and music, reflecting Diego's genuine interests

Background Context

Diego is a technically skilled developer with expertise spanning frontend (React, TypeScript), backend (Python, FastAPI), AI/ML (LangChain, vector databases), and cloud technologies (GCP, AWS, Firebase). He has a passion for music and wants to create a portfolio that reflects both his technical capabilities and creative interests.

Traditional portfolios are often static, boring, and forgettable. Diego wants to create something that recruiters will remember - a portfolio that IS the demonstration of his skills, not just a description of them. By combining his technical expertise with his love for music, he can create a truly unique "Playlist Portfolio" where each section is a "track" in his professional journey, complete with his own music, interactive visualizations, and an AI-powered chatbot "DJ" that can answer questions about his background.

This portfolio will serve as both a creative showcase and a practical hiring tool, providing an immersive experience while ensuring recruiters can quickly find the information they need.

Change Log

Date	Version	Description	Author
2025-01-XX	1.0	Initial PRD creation	John (PM)
2025-01-XX	1.1	Updated per PO validation - Added Story 5.1 details, audio prep task	Sarah (PO)

Requirements

Functional

FR1: The portfolio shall present content organized as five distinct "tracks": University Years, Work Experience, Side Projects, Technical Skills, and Interests & Hobbies.

FR2: The portfolio shall display all tracks in a Spotify-style album view where users can see all tracks simultaneously and click on any track to view its content.

FR3: The portfolio shall include a shuffle mode that allows users to explore tracks in random order.

FR4: The portfolio shall play Diego's original music in the background, with different songs for each track.

FR5: The portfolio shall display audio-reactive visualizations that sync to the music playing in the background.

FR6: The portfolio shall include an AI-powered chatbot "DJ" that can answer questions about Diego's background, skills, projects, education, and interests.

FR7: The AI DJ chatbot shall provide two-tier responses: initial concise summaries with an option to request more detailed information.

FR8: The AI DJ chatbot shall be trained on Diego's resume, projects, university information, coursework, accomplishments, and hobbies using a vector database.

FR9: The portfolio shall provide clickable links to detailed project information and evidence (GitHub repos, live demos, case studies).

FR10: The portfolio shall prominently display contact information and methods to reach Diego.

FR11: Each track shall contain relevant, well-organized content specific to that section (education details, work history, project showcases, skill listings, personal interests).

FR12: The portfolio shall allow users to control audio playback including play, pause, volume adjustment, and mute.

FR13: The portfolio shall maintain a consistent visual aesthetic while allowing each track to express its own mood through subtle visual variations.

Non Functional

NFR1: The portfolio shall be fully responsive and functional on desktop, tablet, and mobile devices.

NFR2: The portfolio shall be accessible to users with disabilities, following WCAG 2.1 AA standards including screen reader compatibility, keyboard navigation, and appropriate color contrast.

NFR3: The portfolio shall load initial content within 3 seconds on standard broadband connections.

NFR4: Audio visualizations shall maintain at least 30 FPS for smooth animations.

NFR5: The AI DJ chatbot shall respond to queries within 2 seconds under normal conditions.

NFR6: The portfolio shall be built using React with Vite for optimal development experience and build performance.

NFR7: The backend AI services shall be implemented using FastAPI with LangChain for AI orchestration.

NFR8: The portfolio shall use Chroma DB or similar vector database for efficient AI knowledge retrieval.

NFR9: The portfolio shall be deployed on a reliable hosting platform (Firebase, Vercel, or AWS) with 99.9% uptime.

NFR10: The codebase shall follow modern best practices with TypeScript for type safety and maintainability.

User Interface Design Goals

Overall UX Vision

The Diego Portfolio creates an immersive, music-themed experience that feels like exploring a personal music streaming platform. The interface should feel familiar to users of Spotify or Apple Music, but with unique touches that showcase Diego's creativity and technical skills. The experience should be primarily lean-back (watching and listening) with strategic interactive moments (clicking for details, chatting with AI DJ).

Key Interaction Paradigms

Primary Navigation:

- Spotify-style album view with all tracks visible
- Click any track to "play" it (view its content)
- Shuffle button for random exploration
- Clear visual indication of current track

Audio Interaction:

- User-initiated playback (no auto-play)
- Persistent audio controls (play/pause, volume, mute)
- Seamless transitions between tracks

AI DJ Interaction:

- Chat interface accessible from any track
- Natural language input
- Conversational responses with personality
- Optional "more details" expansion

Content Discovery:

- Hover effects on interactive elements
- Smooth transitions between tracks
- Click-to-expand for project details
- External links for deeper exploration

Core Screens and Views

1. Landing/Album View

- Overview of all 5 tracks
- Album artwork/hero visual
- Play button to start experience
- Brief introduction or tagline

2. Track View (Individual Pages)

- Track title and description
- Main content for that section
- Audio visualizations
- Navigation to other tracks
- AI DJ chat interface

3. AI DJ Chat Interface

- Chat input field
- Conversation history
- DJ personality/avatar
- Quick action buttons or suggested questions

4. Project Detail Modals/Pages

- Project overview
- Technologies used
- Links to live demo, GitHub, case studies
- Screenshots or demos
- Back to track navigation

Accessibility

Target: WCAG 2.1 AA Compliance

Key Requirements:

- Semantic HTML structure
- Keyboard-only navigation support
- Screen reader announcements for dynamic content
- Sufficient color contrast (4.5:1 minimum)
- Focus indicators on interactive elements
- Alt text for all images
- Captions or transcripts for audio if needed
- Skip navigation links
- Pause/stop controls for animations

Branding

Style:

- Modern, clean, professional yet creative
- Music-inspired aesthetic (album artwork, player controls, track listings)
- Tech-forward visual language
- Consistent with Diego's personal brand

Color Palette:

- TBD during design phase

- Should work well for visualizations
- Must meet accessibility contrast requirements
- Different mood per track while maintaining cohesion

Typography:

- Clean, readable fonts
- Hierarchy that supports content scanning
- Tech-friendly aesthetic

Target Device and Platforms

Primary: Web Responsive (Desktop and Mobile)

- Desktop: Optimal experience with larger visualizations
- Tablet: Adapted layout, full functionality
- Mobile: Simplified but complete experience

Browsers:

- Chrome, Firefox, Safari, Edge (latest 2 versions)
- Progressive enhancement for older browsers

Technical Assumptions

Repository Structure

Type: Monorepo (recommended) or Polyrepo

Rationale: Monorepo allows shared TypeScript types between frontend and backend, easier local development, and coordinated deployments. However, polyrepo is acceptable if preferred for separation of concerns.

Service Architecture

Architecture: Serverless functions for backend (API) + Static hosting for frontend

Components:

- **Frontend:** React SPA hosted on Firebase Hosting, Vercel, or S3+CloudFront
- **Backend API:** FastAPI deployed as serverless functions (Cloud Functions, Lambda, or Vercel Serverless)
- **Vector Database:** Managed Chroma DB or hosted solution

- **Storage:** Cloud storage for audio files and assets

Rationale: Serverless architecture provides cost-effectiveness for a portfolio site with variable traffic, automatic scaling, and simplified deployment.

Testing Requirements

Required Testing:

- Unit tests for critical components and utilities
- Integration tests for AI chatbot functionality
- End-to-end tests for primary user flows
- Manual testing for audio/visual experience

Testing Tools:

- Frontend: Vitest or Jest + React Testing Library
- Backend: pytest for FastAPI
- E2E: Playwright or Cypress

Coverage Goals:

- Minimum 70% code coverage for critical paths
- 100% coverage for AI chatbot logic

Additional Technical Assumptions and Requests

Frontend:

- React 18+ with TypeScript
- Vite for build tooling and dev server
- Web Audio API for audio analysis
- Canvas or WebGL for visualizations (Three.js optional)
- State management: Context API or Zustand (lightweight)
- Styling: TailwindCSS or styled-components (TBD)

Backend:

- Python 3.11+

- FastAPI for API endpoints
- LangChain for AI orchestration
- Chroma DB for vector storage
- OpenAI or similar for embeddings and chat completion

Infrastructure:

- Docker for local development and deployment
- GitHub Actions or similar for CI/CD
- Environment-based configuration
- Secrets management (env variables, cloud secrets manager)

Performance:

- Code splitting for optimal load times
- Lazy loading for track content
- Optimized audio file formats (compressed, streaming-ready)
- CDN for static assets
- Image optimization

Security:

- HTTPS only
- API rate limiting
- Input validation and sanitization
- CORS configuration
- Secure secrets management

Epic List

Epic 1: **Foundation & Project Setup** - Establish project infrastructure, development environment, and basic application structure with routing.

Epic 2: **Album View & Track Navigation** - Implement the core navigation experience with Spotify-style album view and track switching.

Epic 3: **Audio System & Player Controls** - Integrate music playback, audio controls, and basic audio visualizations.

Epic 4: **Track Content & Storytelling** - Populate all five tracks with content, styling, and track-specific designs.

Epic 5: **AI DJ Chatbot** - Build and integrate the AI-powered chatbot with vector database and conversational interface.

Epic 6: **Polish, Performance & Deployment** - Optimize performance, ensure accessibility, conduct testing, and deploy to production.

Epic 1: Foundation & Project Setup

Epic Goal: Establish a solid technical foundation with project scaffolding, development environment, core dependencies, and basic application structure that supports all future features.

Story 1.1: Initialize Project with React + Vite

As a developer,
I want a properly configured React + Vite project with TypeScript,
so that I have a modern, fast development environment.

Acceptance Criteria:

1. Project is initialized with Vite using React + TypeScript template
2. Package.json includes all necessary scripts (dev, build, preview, test)
3. TypeScript configuration is set up with strict mode enabled
4. ESLint and Prettier are configured for code quality
5. Git repository is initialized with appropriate .gitignore
6. README includes project description and setup instructions
7. Development server runs successfully on localhost

Story 1.2: Set Up Project Structure and Routing

As a developer,
I want a well-organized folder structure and routing system,
so that the application is maintainable and scalable.

Acceptance Criteria:

1. Folder structure includes: src/components, src/pages, src/hooks, src/utils, src/styles, src/services
2. React Router is installed and configured
3. Routes are defined for: Home (album view), Track pages (5 routes), 404 page
4. Basic layout component wraps all routes
5. Navigation between routes works correctly
6. Browser back/forward buttons work as expected
7. URL structure is clean and semantic (e.g., /track/university, /track/work)

Story 1.3: Install and Configure Core Dependencies

As a developer,
I want all essential frontend dependencies installed and configured,
so that I can build features efficiently.

Acceptance Criteria:

1. State management solution installed (Context API setup or Zustand)
2. Styling framework installed (TailwindCSS or styled-components)
3. Testing framework installed (Vitest + React Testing Library)
4. HTTP client installed (Axios or Fetch wrapper)
5. All dependencies are compatible versions
6. No conflicting peer dependencies
7. Package lock file is committed

Story 1.4: Create Basic Design System and Theme

As a developer,
I want a foundational design system with colors, typography, and spacing,
so that the UI is consistent across all tracks.

Acceptance Criteria:

1. Theme configuration file defines colors, fonts, spacing scale
2. Global styles are applied (CSS reset, base typography)
3. Reusable style utilities or theme provider is set up

4. Color palette supports light theme (dark mode future consideration)
5. Typography hierarchy is defined (h1-h6, body, small)
6. Spacing scale follows consistent pattern (4px, 8px, 16px, 24px, 32px, etc.)
7. Theme can be easily accessed throughout application

Story 1.5: Set Up Backend Project Structure

As a developer,
I want a FastAPI backend project initialized,
so that I can build the AI DJ chatbot API.

Acceptance Criteria:

1. Python virtual environment is created
2. FastAPI is installed with required dependencies (uvicorn, pydantic)
3. Project structure includes: api/routes, api/services, api/models, api/utils
4. Basic FastAPI app runs successfully
5. Health check endpoint returns 200 OK
6. CORS middleware is configured for local development
7. Environment variable loading is configured (.env support)

Epic 2: Album View & Track Navigation

Epic Goal: Build the core Spotify-style navigation experience where users can see all tracks, click any track to view it, and use shuffle mode for random exploration.

Story 2.1: Create Album View Landing Page

As a visitor,
I want to see an album-style overview of all five tracks,
so that I can understand the portfolio structure and choose where to explore.

Acceptance Criteria:

1. Landing page displays album artwork or hero visual
2. All five tracks are displayed as a list with track numbers and titles
3. Each track shows a brief one-line description

4. Visual design resembles Spotify album view
5. Page is responsive on desktop, tablet, and mobile
6. "Play" or "Start" button begins the experience
7. Portfolio title "Diego Portfolio" is prominently displayed

Story 2.2: Implement Track Selection and Navigation

As a visitor,
I want to click on any track to view its content,
so that I can explore Diego's portfolio in any order I choose.

Acceptance Criteria:

1. Clicking a track navigates to that track's dedicated page
2. URL updates to reflect current track (e.g., /track/university)
3. Track navigation preserves application state
4. Current track is visually indicated in navigation
5. Users can return to album view from any track
6. Track transitions are smooth
7. Deep linking works (sharing a track URL loads that track directly)

Story 2.3: Build Track Navigation Controls

As a visitor,
I want navigation controls to move between tracks,
so that I can easily browse through Diego's story sequentially or randomly.

Acceptance Criteria:

1. "Previous Track" and "Next Track" buttons are available on each track page
2. Previous/Next buttons loop (Track 5 → Track 1 when clicking next)
3. Track list sidebar or menu shows all tracks with current track highlighted
4. Clicking a track in the sidebar navigates to that track
5. Keyboard shortcuts work (arrow keys for prev/next)
6. Navigation controls are accessible via keyboard
7. Mobile navigation is optimized (hamburger menu or swipe gestures)

Story 2.4: Implement Shuffle Mode

As a visitor,
I want a shuffle button that takes me to a random track,
so that I can explore the portfolio in an unexpected, playful way.

Acceptance Criteria:

1. Shuffle button is visible and accessible from album view and track pages
2. Clicking shuffle navigates to a random track
3. Shuffle excludes the current track (doesn't reload same track)
4. Shuffle button has appropriate icon (shuffle/random icon)
5. Visual feedback indicates shuffle action occurred
6. Shuffle works correctly on all devices
7. Shuffle mode can be toggled on/off for continuous random navigation (optional enhancement)

Story 2.5: Create Track Page Template

As a developer,
I want a reusable track page component,
so that all five tracks have consistent structure while allowing content variation.

Acceptance Criteria:

1. Track page component accepts props for title, content, and styling
2. Layout includes: track header, content area, navigation controls, audio controls area
3. Each track can define its own mood/theme through style props
4. Component is responsive across all screen sizes
5. Accessibility features are built in (semantic HTML, ARIA labels)
6. Loading states are handled gracefully
7. Component is tested with unit tests

Epic 3: Audio System & Player Controls

Epic Goal: Integrate music playback capabilities with Diego's original songs, audio controls, and real-time audio-reactive visualizations that create an immersive experience.

Story 3.1: Implement Audio Player with Playback Controls

As a visitor,
I want to control music playback (play, pause, volume, mute),
so that I can manage my audio experience according to my preferences.

Acceptance Criteria:

1. Audio player component loads and plays MP3 files
2. Play/Pause button toggles audio playback
3. Volume slider controls audio volume (0-100%)
4. Mute button toggles audio on/off
5. Audio controls are always accessible (persistent player bar)
6. Current playback time and total duration are displayed
7. Seek bar allows jumping to specific time in track
8. Audio controls are keyboard accessible
9. Mobile controls are touch-optimized

Story 3.2: Set Up Music Files and Track Audio Association

As a developer,
I want each track to have its own associated music file,
so that different music plays when visiting different sections of the portfolio.

Acceptance Criteria:

1. Audio files prepared and optimized for web delivery (see prerequisite task below)
2. Audio files stored in appropriate directory (public/audio or cloud storage)
3. Each track (University, Work, Projects, Skills, Hobbies) has a designated music file
4. Music automatically loads when navigating to a track
5. Music crossfades or transitions smoothly between tracks (optional enhancement)
6. Loading states display while audio files load
7. Error handling for failed audio loads

Prerequisite Task - Audio File Preparation: Before implementing this story, prepare audio files:

- [ ] Create or select 5 original music tracks (one per portfolio section)
- [ ] Each track should be 2-4 minutes in length
- [ ] Compress to MP3 format at 128-192kbps for optimal web delivery
- [ ] Normalize volume levels across all tracks for consistency
- [ ] Name files clearly: `track-1-university.mp3`, `track-2-work.mp3`, etc.
- [ ] Test playback in browser to ensure quality
- [ ] Total audio file size should be < 15MB combined

Story 3.3: Implement Web Audio API Integration

As a developer,
I want to use Web Audio API for audio analysis,
so that I can create data-driven visualizations that respond to the music.

Acceptance Criteria:

1. Web Audio API context is created and managed
2. Audio source is connected to analyser node
3. Frequency data is extracted in real-time
4. Time domain data is available for waveform visualization
5. Audio analysis doesn't impact playback quality
6. Browser compatibility is handled gracefully
7. Audio context is properly cleaned up when unmounting

Story 3.4: Create Basic Audio Visualizations

As a visitor,
I want to see dynamic visualizations that respond to the music,
so that the experience feels alive and immersive.

Acceptance Criteria:

1. Canvas element renders visualization in real-time
2. Visualization reacts to audio frequency data
3. Visualization maintains at least 30 FPS
4. Visual style matches track aesthetic
5. Visualization is responsive to different screen sizes

6. Visualization pauses when audio is paused
7. Multiple visualization styles can be selected or randomized (bars, waveform, particles)
8. Visualizations are performant and don't cause UI lag

Story 3.5: Implement Audio State Management

As a developer,
I want centralized audio state management,
so that audio persists correctly across navigation and components can react to audio state.

Acceptance Criteria:

1. Audio state (playing, paused, current time, volume) is globally accessible
2. Audio continues playing when navigating between tracks (if desired)
3. Audio state persists across page refreshes (saved to localStorage)
4. Multiple components can read and control audio state
5. Audio state updates trigger appropriate UI updates
6. No memory leaks or duplicate audio instances
7. Audio automatically stops/cleans up on page unload

Epic 4: Track Content & Storytelling

Epic Goal: Populate all five tracks with compelling content, styling, and unique moods that tell Diego's professional story while maintaining visual cohesion.

Story 4.1: Build Track 1 - University Years

As a visitor,
I want to learn about Diego's educational background,
so that I understand his academic foundation and learning journey.

Acceptance Criteria:

1. Track displays university name, degree, graduation year
2. Key coursework is listed with brief descriptions
3. Academic achievements are highlighted
4. Relevant university projects are showcased

5. Content is well-organized with clear sections
6. Visual mood reflects "foundation-building" theme
7. Links to certificates or transcripts (if applicable)

Story 4.2: Build Track 2 - Work Experience

As a visitor,
I want to see Diego's professional work history,
so that I can understand his career progression and experience.

Acceptance Criteria:

1. Each position displays: company, role, dates, location
2. Key responsibilities are listed with bullet points
3. Notable achievements are highlighted with metrics when possible
4. Career progression is chronologically clear
5. Visual mood reflects "professional accomplishment" theme
6. Company logos are displayed (if available)
7. Links to company websites or recommendations

Story 4.3: Build Track 3 - Side Projects

As a visitor,
I want to explore Diego's personal and side projects,
so that I can see his passion, creativity, and initiative beyond work.

Acceptance Criteria:

1. Each project has: title, description, tech stack, and links
2. Projects are displayed as cards or list items
3. Clickable projects open detailed modal or page
4. GitHub links, live demos, and case studies are accessible
5. Project images or screenshots are displayed
6. Visual mood reflects "creative and experimental" theme
7. Projects can be filtered by technology or category (optional)

Story 4.4: Build Track 4 - Technical Skills

As a visitor,
I want to see Diego's comprehensive technical skill set,
so that I can quickly assess if his skills match my needs.

Acceptance Criteria:

1. Skills are organized by category (Languages, Frameworks, AI/ML, Cloud, etc.)
2. Skill categories match: Languages, AI/ML & Data, Frameworks/Tools, Cloud & DevOps
3. Each skill has visual representation (logo, proficiency level optional)
4. Skills are scannable and visually organized
5. Visual mood reflects "technical expertise" theme
6. Skills can be clicked for more context (years of experience, projects used in)
7. Layout is responsive and accessible

Story 4.5: Build Track 5 - Interests & Hobbies

As a visitor,
I want to learn about Diego's personal interests,
so that I can connect with him as a person beyond his technical skills.

Acceptance Criteria:

1. Personal interests are listed with descriptions
2. Music passion is prominently featured
3. Hobbies are showcased authentically
4. Content shows personality and humanity
5. Visual mood reflects "personal and authentic" theme
6. Photos or media enhance the storytelling (optional)
7. Balance between personal and professional tone

Story 4.6: Add Project Detail Views

As a visitor,
I want to click on a project to see detailed information,
so that I can understand the project deeply without cluttering the main track view.

Acceptance Criteria:

1. Clicking a project opens modal or dedicated page
2. Detail view includes: full description, problem statement, solution approach
3. Technologies used are clearly listed
4. Links to GitHub, live demo, case study are prominent
5. Screenshots, GIFs, or videos showcase the project
6. User can close detail view and return to track
7. Detail views are accessible via keyboard
8. Mobile experience is optimized

Epic 5: AI DJ Chatbot

Epic Goal: Build and integrate an intelligent AI-powered chatbot that serves as a "DJ" guide, answering questions about Diego's background using vector database retrieval and natural language processing.

Story 5.1: Set Up and Populate Vector Database

As a developer,
I want to set up Pinecone with Diego's information embedded,
so that the AI can retrieve relevant context to answer questions.

Acceptance Criteria:

1. Pinecone account created and API key obtained
2. Pinecone index created with correct dimensions (1536 for OpenAI embeddings)
3. Training data collected and prepared: resume, projects, university info, coursework, skills, interests, hobbies
4. Training data is chunked appropriately (max 500 tokens, 50 token overlap)
5. Knowledge base documents organized by category (education, work, projects, skills, interests)
6. Embeddings generated using OpenAI text-embedding-3-small
7. Embeddings stored in Pinecone with proper metadata (text, source, category, date)
8. Vector similarity search tested and returns relevant documents
9. Ingest script (`ingest_knowledge.py`) created for future updates

10. Documentation added explaining how to update knowledge base

Story 5.2: Build FastAPI Chatbot Endpoint

As a developer,
I want a FastAPI endpoint that processes chat queries,
so that the frontend can communicate with the AI chatbot.

Acceptance Criteria:

1. POST endpoint accepts user messages and conversation history
2. Endpoint retrieves relevant context from vector database
3. Endpoint calls LLM (OpenAI/Anthropic) with context and query
4. Response is generated with friendly, casual, fun personality
5. Endpoint returns response with proper error handling
6. API response time is under 2 seconds
7. Rate limiting prevents abuse
8. CORS is configured for frontend access

Story 5.3: Implement Chatbot Frontend UI

As a visitor,
I want a chat interface to talk with the AI DJ,
so that I can ask questions about Diego and get immediate answers.

Acceptance Criteria:

1. Chat interface is accessible from all track pages
2. Chat icon/button is prominently displayed
3. Chat window opens with welcoming message
4. User can type messages and send them
5. AI responses appear in conversation thread
6. Chat interface is styled to match portfolio aesthetic
7. Chat is accessible via keyboard
8. Mobile chat interface is optimized
9. Chat conversation history persists during session

Story 5.4: Implement Two-Tier Response Pattern

As a visitor,
I want to receive concise answers first with an option to get more details,
so that I can control the depth of information I receive.

Acceptance Criteria:

1. Initial AI responses are concise summaries (2-3 sentences)
2. "Want more details?" or "Tell me more" button appears after each response
3. Clicking for more details provides expanded information
4. Expanded responses include additional context and examples
5. User can continue asking follow-up questions naturally
6. Conversation flow feels natural and not overly structured
7. Details expansion works correctly on mobile

Story 5.5: Add Conversation Context and Suggested Questions

As a visitor,
I want the chatbot to remember our conversation and suggest relevant questions,
so that the experience feels intelligent and helpful.

Acceptance Criteria:

1. Chatbot maintains conversation history within session
2. Follow-up questions understand previous context
3. Suggested questions appear based on conversation topic
4. Suggested questions are clickable to auto-send
5. At least 3-5 suggested questions are available
6. Suggestions update based on conversation flow
7. Conversation can be reset/cleared by user

Story 5.6: Add AI DJ Personality and Prompting

As a developer,
I want the AI chatbot to have a consistent friendly, casual, fun personality,
so that interactions feel personal and aligned with the "DJ" theme.

Acceptance Criteria:

1. System prompt defines AI DJ personality traits
2. AI uses casual, friendly language consistently
3. AI occasionally uses music/DJ-related metaphors naturally
4. AI avoids overly formal or robotic language
5. AI stays on-topic about Diego and doesn't hallucinate information
6. AI gracefully handles questions it can't answer
7. Personality testing confirms consistent tone across diverse questions

---

Epic 6: Polish, Performance & Deployment

Epic Goal: Optimize the portfolio for performance, ensure full accessibility compliance, conduct comprehensive testing, and deploy to production hosting.

Story 6.1: Optimize Performance and Loading Times

As a visitor,
I want the portfolio to load quickly and run smoothly,
so that I have a seamless experience without lag or delays.

Acceptance Criteria:

1. Initial page load is under 3 seconds on broadband
2. Images are optimized and lazy-loaded
3. Audio files are compressed and streamable
4. Code splitting reduces initial bundle size
5. Unused dependencies are removed
6. Lighthouse performance score is 90+
7. Audio visualizations maintain 30+ FPS
8. No console errors or warnings in production

Story 6.2: Ensure Full Accessibility Compliance

As a visitor with disabilities,
I want the portfolio to be fully accessible,

so that I can experience Diego's portfolio regardless of my abilities.

Acceptance Criteria:

1. All interactive elements are keyboard navigable
2. Focus indicators are visible and clear
3. Screen readers can access all content meaningfully
4. Color contrast meets WCAG AA standards (4.5:1 minimum)
5. All images have appropriate alt text
6. ARIA labels and roles are properly implemented
7. Skip navigation links are available
8. Form inputs have associated labels
9. Dynamic content changes are announced to screen readers
10. Accessibility testing with axe DevTools shows no violations

Story 6.3: Implement Responsive Design for All Devices

As a visitor on any device,
I want the portfolio to work beautifully on my screen size,
so that I have an optimal experience whether on phone, tablet, or desktop.

Acceptance Criteria:

1. Portfolio is fully functional on mobile (320px+), tablet (768px+), and desktop (1024px+)
2. Touch interactions work correctly on mobile and tablet
3. Navigation adapts to mobile (hamburger menu or simplified nav)
4. Audio controls are touch-optimized
5. Visualizations scale appropriately to screen size
6. Text is readable without zooming on all devices
7. No horizontal scrolling occurs
8. Images and media scale proportionally

Story 6.4: Write Comprehensive Tests

As a developer,

I want comprehensive test coverage,
so that I can confidently deploy and maintain the portfolio.

Acceptance Criteria:

1. Unit tests cover critical components (audio player, navigation, track components)
2. Integration tests verify AI chatbot functionality
3. E2E tests cover primary user flows (navigate tracks, play audio, use chat)
4. Test coverage is at least 70% for critical paths
5. All tests pass in CI pipeline
6. Tests run automatically on git push
7. Test documentation explains how to run tests locally

Story 6.5: Deploy Backend API to Production

As a developer,
I want the FastAPI backend deployed to production hosting,
so that the AI chatbot is available to portfolio visitors.

Acceptance Criteria:

1. Backend API is deployed to chosen platform (Cloud Functions, Lambda, or Vercel)
2. Environment variables are configured securely
3. API endpoints are accessible via HTTPS
4. CORS is configured for production frontend domain
5. Health check endpoint returns successfully
6. API rate limiting is enabled
7. Monitoring and logging are set up
8. Deployment is automated via CI/CD

Story 6.6: Deploy Frontend to Production Hosting

As a developer,
I want the React frontend deployed to production,
so that Diego's portfolio is publicly accessible.

Acceptance Criteria:

1. Frontend is deployed to chosen platform (Firebase Hosting, Vercel, or S3+CloudFront)
2. Custom domain is configured (optional)
3. HTTPS is enabled
4. Environment variables point to production API
5. Build optimizations are enabled
6. CDN caching is configured appropriately
7. Deployment is automated via CI/CD
8. Portfolio is accessible and functional at production URL

Story 6.7: Add Analytics and Monitoring

As a portfolio owner,
I want to track visitor behavior and monitor application health,
so that I can understand how recruiters interact with my portfolio and catch issues quickly.

Acceptance Criteria:

1. Analytics tool is integrated (Google Analytics, Plausible, or similar)
2. Key events are tracked (page views, track navigation, chatbot usage, project clicks)
3. Error tracking is set up (Sentry or similar)
4. Performance monitoring captures Core Web Vitals
5. API error rates and response times are monitored
6. Privacy policy is added if required by analytics tool
7. Analytics dashboard is accessible to Diego

Checklist Results Report

To be populated after running pm-checklist

Next Steps

UX Expert Prompt

"I've completed the PRD for Diego Portfolio - a unique music-themed portfolio website with 5 tracks, audio visualizations, and an AI DJ chatbot. Please review the User Interface Design Goals section and create a comprehensive UI/UX specification document that details:

- Detailed wireframes or mockups for the album view and track pages
- Visual design system (colors, typography, spacing)
- Component specifications for audio player, navigation, and chat interface
- Accessibility implementation details
- Responsive breakpoints and mobile adaptations

Use the front-end-spec-tmpl.yaml template to create the specification."

Architect Prompt

"I've completed the PRD for Diego Portfolio. Please review the entire PRD, especially the Technical Assumptions section, and create a comprehensive architecture document that defines:

- Detailed technical architecture for frontend (React + Vite)
- Backend API architecture (FastAPI + LangChain + Chroma DB)
- Audio system implementation using Web Audio API
- Vector database setup and embedding strategy
- Deployment architecture (serverless functions + static hosting)
- Data models for projects, tracks, and chat conversations
- API specifications for chatbot endpoints
- Infrastructure and CI/CD pipeline

Use the architecture-tmpl.yaml or fullstack-architecture-tmpl.yaml template to create the architecture document."