AI 7993 – Section W01 – Spring 2026

[Indy-1]- NextMission Navigator

Crystal Tubbs

April 27, 2026

Website: www.VetNavi.AI

GitHub: github.com/Msmetamorphosis (Private – Potential IP)

Presentation video: https://youtu.be/k1pcMwI1snM

Project website: https://msmetamorphosis.github.io/nextmission-capstone/

Table of Contents

1. Introduction

NextMission Navigator (VetNavi) is a full-stack, AI-powered web application built as the capstone project for AI 7993. The platform assists U.S. military veterans transitioning to civilian life by generating personalized, goal-specific action plans using natural language input, curated regional resource datasets, and voice AI integration.

This is a complete from-scratch rebuild (v3) of a prior prototype, replacing a low-code implementation with a production-grade custom-coded solution using Next.js 15, React 19, TypeScript, and the Anthropic Claude API. The project is live at www.VetNavi.AI.

Problem Statement

Over 200,000 service members transition out of the military each year. Many face an overwhelming, fragmented landscape of benefits, career resources, housing options, and healthcare services. Traditional systems do not provide personalized guidance tailored to a veteran's unique background, location, and goals. NextMission Navigator bridges this gap using AI-driven action planning, regional resource injection, and voice AI accessibility.

Target Users

- U.S. military veterans transitioning from active duty to civilian life
- Transitioning service members approaching separation
- Military spouses assisting with transition planning
- Workforce support coordinators using the platform as a guidance tool

Project Scope

The platform generates structured, step-by-step action plans tailored to individual veteran goals including career transition, education planning, housing acquisition, disability claims, and benefits navigation. A RAG architecture grounds LLM outputs in verified regional resource datasets. An AI chatbot and ElevenLabs voice agent provide 24/7 interactive support.

2. Requirements Analysis

Full requirements were documented in the Software Requirements Specification (SRS v1.0). Key functional and non-functional requirements are summarized below.

Functional Requirements

Component	Details
REQ-IE-1.1	Accept structured inputs: goal, military branch, years of service, location, target industry
REQ-IE-1.2	Generate structured action plans with prioritized steps, categories, timeframes, and resource links
REQ-IE-1.3	Validate all LLM outputs against predefined ActionPlan Zod schema before returning results
REQ-IE-1.4	Reject or repair schema-non-conforming outputs; fall back to mock plans if needed
REQ-RRI-1.1	Maintain curated regional resource datasets indexed by geography and category
REQ-RRI-1.4	Inject validated regional data into LLM generation context (RAG architecture)
REQ-VIM-1.1	Support text-to-speech conversion via ElevenLabs ConvAI widget
REQ-VIM-1.4	Handle voice service unavailability gracefully without impacting core plan generation

Non-Functional Requirements

Component	Details
Performance	Plan generation target under 15 seconds; stateless API enables horizontal scaling
Security	API keys stored in server-side environment variables only; HTTPS for all communications
Safety	Prominent Veterans Crisis Line (988) displayed on all pages; disclaimers on all AI outputs
Usability	WCAG 2.1 AA accessibility; voice AI support; clear, non-technical error messaging
Reliability	Fallback mock plans ensure users always receive guidance even during API failures

Key Design Constraints

- All AI API keys must remain server-side and never exposed to client code
- Generated plans must be informational only – not legal, medical, or financial advice
- Curated regional datasets must be manually verified before deployment
- The system must degrade gracefully when external services (Claude API, ElevenLabs) are unavailable

3. Design

System Architecture

NextMission Navigator uses a layered, stateless serverless architecture deployed on Vercel. The five primary subsystems are:

Component	Details
Frontend (Next.js 15)	React 19 App Router pages: Home, Resources, Veterans, About, Contact. Military-themed design system with Tailwind CSS and custom CSS variables.
Backend API Layer	Next.js API routes at /api/action-plan and /api/chat. Handles Zod schema validation, goal classification, resource hint injection, and LLM orchestration.
Intelligence Engine	Anthropic Claude API wrapper (claude-3-5-sonnet-20241022). Constructs structured prompts, validates JSON responses, and falls back to mock plans on failure.
Knowledge Curation Layer	TypeScript resource catalog (catalog.ts) with 20+ verified resources across 6 categories. Regional matching supports national, state (TX, FL), and metro-level resources.
Voice Module	ElevenLabs ConvAI widget (embedded web component). Separate AI chat endpoint at /api/chat for text-based conversation. Voice typing via Web Speech API in the chatbot component.

Data Flow: Action Plan Generation

The following describes the complete request lifecycle for action plan generation:

1. User submits goal + optional context (branch, years of service, location, target industry) via ActionPlanGenerator component.
2. Frontend POSTs to /api/action-plan with Zod-validated body (goal: min 5 chars, max 8000).
3. Backend classifies goal into one of 7 categories (Housing, Disability, Healthcare, Education, Career, Financial, General) using keyword matching in goalClassify.ts.
4. resourceMatch.ts queries the curated catalog, scoring resources by keyword relevance and regional match. Up to 4 resource hints are injected into the LLM prompt.
5. generateActionPlan.ts constructs the full prompt combining system instructions, user context, goal classification, and resource hints.
6. Anthropic Messages API is called with a 55-second timeout. The response JSON is extracted and normalized via actionPlanOutput.ts Zod schema.
7. attachCatalogNotes() appends verified catalog links to the first step of the returned plan.
8. The validated ActionPlanResponse is returned to the frontend and rendered as categorized step cards with priority badges, timeframes, and resource links.

Key Architectural Decisions

- Stateless API design: No session or database storage in v1.0 – all context passed per-request, enabling automatic horizontal scaling on Vercel.
- Zod schema validation: Both input (request body) and output (LLM response) are validated. Malformed LLM responses trigger fallback to mock plans, ensuring users always receive a response.
- RAG via catalog injection: Regional resource hints are injected directly into the LLM prompt rather than using a vector database, keeping the architecture lightweight and avoiding hallucinated resource recommendations.
- Server-side API key isolation: The anthropic.ts wrapper retrieves ANTHROPIC_API_KEY from process.env at runtime. Keys are never bundled into client-side code.
- Graceful degradation: If the Anthropic API key is absent or the API call fails, mockPlans.ts returns a contextually appropriate fallback plan so the UI never shows a blank error.

UI Design System

The interface uses a military-themed color palette defined as CSS variables in globals.css, including deep army green (#333C2C), sandstone beige (#C2BA9A), coyote tan (#A37E4C), and muted gold (#BCA55A). Component patterns include card-hover lift effects, gradient backgrounds, and military texture overlays. The design follows WCAG 2.1 AA contrast requirements.

4. Development

Note: The source code repository (github.com/Msmetamorphosis/nextmission-vetnavi-v3) is private as this project represents potentially protected intellectual property developed under Metamorphic Curations LLC. Key implementation details are documented below.

Technology Stack

Component	Details
Framework	Next.js 15.5 (App Router), React 19, TypeScript 5.9
Styling	Tailwind CSS 3.4 + custom CSS variable design system
AI / LLM	Anthropic Claude API (claude-3-5-sonnet-20241022, temp 0.7, max_tokens 3000)
Schema Validation	Zod 3.25 – input validation and LLM output normalization
Voice AI	ElevenLabs ConvAI web component (agent: agent_01jysdpp5wehx800c3a7jtfwz3)
Icons	Lucide React 0.468
Deployment	Vercel (serverless edge functions, CDN global distribution)
Build Tools	PostCSS, Autoprefixer, ESLint (eslint-config-next)

Core Implementation: Intelligence Engine

The intelligence engine is implemented across four server-side TypeScript modules:

anthropic.ts – API Wrapper

A minimal, dependency-free wrapper around the Anthropic Messages REST API. Uses native fetch with structured error handling. Accepts an AbortSignal for timeout control. Returns the first text content block from the response.

goalClassify.ts – Goal Classification

Classifies user goals into 7 categories using keyword matching: Housing, Disability Benefits, Healthcare, Education, Career, Financial, and General Transition. Each category returns a label, matched keywords, and a planner blurb that is injected into the LLM prompt to focus generation.

generateActionPlan.ts – Orchestration Layer

Builds the complete user prompt by combining goal text, goal classification, veteran context (branch, years served, location, target industry), and up to 4 curated resource hints from the catalog. Calls the Anthropic API with a 55-second abort controller. Parses and normalizes the

JSON response via Zod. On any failure, falls back to mockPlans.ts. Attaches catalog resource notes to the first step of every plan.

actionPlanOutput.ts – Schema & Normalization

Defines the ActionPlanResponse Zod schema: why_this_plan (optional string), categories (array of named step groups), follow_up (string). Each step contains title, description, optional link (markdown format), timeframe, priority (high/medium/low), and additionalInfo. The normalizePlan() function coerces priorities to lowercase and sets default timeframes. extractJsonObject() strips markdown code fences and extracts raw JSON from LLM text output.

Core Implementation: Knowledge Curation

The resource catalog (src/data/resources/catalog.ts) contains 20+ verified resources organized into 6 categories: Career Transition, Education Benefits, Housing Assistance, Healthcare & Wellness, Financial Support, and Community & Support. Each resource includes name, URL, description, type (Government/Organization/Program), tags for keyword matching, and regions (ALL, TN, GA, TX, FL, TAMPA).

resourceMatch.ts implements region-aware resource scoring. It normalizes user location strings to region tokens (e.g., 'Macdill' maps to FL/FLORIDA/TAMPA tokens), scores resources by keyword overlap with the user's goal, boosts regional resources over national ones, deduplicates by URL, and returns the top 4 matches. This approach ensures Tampa-area veterans see James A. Haley VA Medical Center while Texas veterans see the Texas Veterans Land Board.

Core Implementation: Frontend Components

ActionPlanGenerator.jsx

The primary interactive component. Manages goal text, four optional context fields (branch, years of service, location, target industry), loading state, and plan display. Includes 5 clickable example prompts. Parses markdown-formatted links from step.link fields using a regex helper. Implements a follow-up refinement loop that appends user additions to the original goal and regenerates the plan. Renders steps as categorized card groups with priority color-coding and timeframe display.

Chatbot.jsx

A floating chat widget with full conversation history. Sends message history to /api/chat on each turn. Implements browser Web Speech API voice input (SpeechRecognition) with graceful fallback for unsupported browsers. Auto-scrolls to latest message. Shows animated typing indicator during API calls. The CHAT_SYSTEM prompt (chat.ts) keeps responses under 180 words and always surfaces the Veterans Crisis Line (988) for urgent mental health queries.

ElevenLabsWidget.jsx

Dynamically loads the ElevenLabs ConvAI web component script from unpkg CDN on mount (avoiding duplicate script injection). Renders the elevenlabs-convai custom element with the configured agent ID. Positioned as a fixed overlay via CSS, separate from the text chatbot to provide two distinct interaction modalities.

Resource Catalog UI (ResourcesAccordions.tsx)

An accordion-based resource directory with 6 category sections. Each category uses a distinct background color matching the design system. Resources render as two-column card grids with type badges (Government/Organization/Program), descriptions, and external links. The Veterans Crisis Line is displayed in a prominent red alert banner at the top of the resources page.

API Routes

Component	Details
POST /api/action-plan	Validates body with Zod (goal min 5 / max 8000 chars, optional userContext fields). Calls generateActionPlan(). Returns ActionPlanResponse JSON.
POST /api/chat	Validates message history array (1-30 messages, each max 12000 chars, roles user/assistant). Calls generateChatReply(). Returns { message: string }.

Prompt Engineering

The ACTION_PLAN_SYSTEM prompt (src/prompts/actionPlan.ts) enforces strict JSON-only output with no markdown fences. It defines the exact response schema inline, requires 3-5 steps across categories, mandates https markdown links, and instructs the model to produce a follow-up question. The prompt is intentionally short to maximize schema adherence. The CHAT_SYSTEM prompt (chat.ts) enforces a 180-word response limit and mandates crisis line inclusion for urgent situations.

5. Version Control Summary

All development work was managed using Git hosted on GitHub (github.com/Msmetamorphosis). The repository is private as this project represents potentially protected intellectual property. The branching strategy and commit practices are documented below.

Branching Strategy

Component	Details
main	Stable production-ready branch. Only merged after testing complete features.
dev	Primary development branch for ongoing integration work.
feature/*	Separate branches per feature (e.g., feature/voice-integration, feature/resource-catalog, feature/chat-endpoint).
bugfix/*	Hotfix branches for issue resolution.

Commit Practices

- Atomic commits with descriptive messages following <type>: <description> convention (e.g., feat: add Zod schema validation to action-plan route, fix: handle missing API key with mock fallback).
- Frequent commits at logical breakpoints to maintain a clear development history aligned with milestone deliverables.
- Semantic version tagging at major milestones: v1.0-alpha (prototype), v1.0-beta (feature complete), v1.0-final (C-Day submission).

Key Milestones in Version History

- Milestone 1 (Feb 7): Repository setup, tech stack finalized, initial Next.js 15 scaffold with Tailwind design system.
- Milestone 2 (Feb 28): Anthropic API wrapper, Zod schema validation, /api/action-plan route, goal classification engine.
- Milestone 3 (Mar 15): Resource catalog with regional matching, RAG prompt injection, ElevenLabs ConvAI widget, /api/chat route.
- Milestone 4 (Mar 30): ActionPlanGenerator UI with follow-up loop, Chatbot with voice input, ResourcesAccordions, full Veterans and About pages.
- Milestone 5 (Apr 15): Production deployment to Vercel, final UI polish, mock plan fallbacks, crisis line integration on all pages.

6. Testing

Test Approach

Testing followed a practical integration-first approach given the solo development context and the AI-centric nature of the system. Formal unit testing was scoped as future work per the project plan; testing focused on functional validation, schema compliance verification, and real-world scenario testing.

Schema Compliance Testing

The Zod schema validation layer in actionPlanOutput.ts was manually verified against multiple LLM response variations including missing fields, incorrect types, nested category structures, and empty steps arrays. The normalizePlan() function was confirmed to correctly coerce priority strings and apply default timeframes. The extractJsonObject() function was tested against LLM outputs containing markdown code fences, prose before/after JSON, and responses with no JSON at all.

API Endpoint Testing

Both API routes were tested via direct HTTP requests using browser DevTools and manual fetch calls:

- /api/action-plan: Verified Zod rejection of goals under 5 characters, goals over 8000 characters, and invalid userContext shapes. Verified successful plan generation across all 7 goal categories. Verified mock plan fallback when ANTHROPIC_API_KEY is absent.
- /api/chat: Verified rejection of empty message arrays, messages exceeding 30 turns, and content exceeding 12000 characters. Verified multi-turn conversation continuity.

Regional Resource Matching Testing

The pickResourceHints() function was tested with location strings including ZIP codes, city names, military base names, and empty strings. Confirmed that 'Fort Liberty, NC' does not trigger Florida resources, that 'Tampa' correctly surfaces James A. Haley VA Medical Center, and that national resources (regions: ['ALL']) appear for all locations.

End-to-End Scenario Testing

Five representative veteran scenarios were tested end-to-end against the live production deployment:

Component	Details
Cybersecurity Career (Army, NC)	Correct career category classification; Security+ and TAP resources surfaced; 5-step plan generated in under 8 seconds.

VA Disability Claim (Navy, FL)	Disability category detected; VA.gov disability link included; James A. Haley VA highlighted for Tampa location.
GI Bill Education (Air Force, TX)	Education category; Yellow Ribbon and GI Bill Comparison Tool included; Texas Veterans Land Board surfaced for TX location.
VA Home Loan (Marines, general)	Housing category; VA Home Loan eligibility step as first action; COE and lender comparison steps included.
Mental Health Support (general)	Healthcare category; Veterans Crisis Line 988 prominently included in step with high priority; VA healthcare enrollment as next step.

Voice and Chat Testing

The ElevenLabs ConvAI widget was verified to load correctly across Chrome, Safari, and Edge. The text chatbot was tested for multi-turn continuity, voice input transcription accuracy (Chrome SpeechRecognition API), and graceful handling of API timeouts. Crisis line surfacing was verified for queries containing keywords related to mental health distress.

7. Challenges, Assumptions & Risk Assessment

Development Challenges

LLM Output Reliability

The most significant technical challenge was enforcing structured JSON output from the Anthropic Claude API. Early iterations produced inconsistent results when the model prepended prose explanations or wrapped JSON in markdown code fences. This was resolved by keeping the system prompt minimal and explicit, defining the JSON schema directly in the prompt, and implementing the extractJsonObject() function to strip fences and locate JSON anywhere in the response text. A complete mock plan fallback was added as the final reliability layer.

Regional Resource Matching Without a Vector Database

Implementing meaningful RAG without ChromaDB or a vector database required designing a custom keyword and region scoring system. The solution uses tag-based keyword overlap scoring combined with explicit region token matching. While less sophisticated than embedding-based retrieval, this approach is transparent, fully typed, deterministic, and requires no external dependencies or database management.

Next.js 15 / React 19 Compatibility

Several component patterns from earlier React versions required updates for React 19 strict mode and the Next.js 15 App Router. The ElevenLabs web component required using React's createElement() to render the custom HTML element rather than JSX, as JSX does not natively support arbitrary custom element attributes like agent-id.

Key Assumptions

- Users have reliable internet access and a modern browser (Chrome, Safari, Edge, or Firefox).
- The Anthropic Claude API and ElevenLabs services maintain uptime consistent with their SLA commitments.
- Curated resource URLs remain valid; quarterly manual re-verification is assumed as an operational responsibility.
- Veterans will provide sufficiently specific goals to enable meaningful plan generation; the follow-up question mechanism handles under-specified inputs.

Risk Assessment Summary

Component	Details
API Downtime / Rate Limits (Medium / High)	Mock plan fallback; graceful error messaging; AbortController timeout protection.
LLM Schema Failures (Medium / Medium)	Zod validation + normalizePlan() coercion + extractJsonObject() fence stripping + mock fallback.
Security Vulnerabilities (Low / High)	Server-side only API keys; no client-side key exposure; Zod input validation prevents injection.
Voice Integration Issues (Medium / Medium)	Text plan always accessible; voice is additive enhancement only; ElevenLabs errors non-blocking.
Scope Creep / Timeline (Medium / High)	Weekly self-review checkpoints; core features prioritized; community/auth features deferred to v2.
Stale Resource Data (Low / Medium)	Curated catalog is small and manually maintained; quarterly review planned for production.

8. Conclusion / Summary

NextMission Navigator v3 successfully delivers a production-grade, AI-powered veteran transition platform that meets or exceeds all core capstone requirements. The system is live at www.VetNavi.AI and demonstrates the practical application of large language models, retrieval-augmented generation, schema-enforced output validation, and voice AI integration in a meaningful real-world context.

The rebuild from a low-code prototype to a fully custom TypeScript codebase resulted in a significantly more reliable, maintainable, and extensible system. The Zod-based dual-layer validation (input and output) proved essential for consistent AI output quality. The lightweight RAG implementation using a typed resource catalog and keyword scoring provides locally-relevant guidance without the operational overhead of a vector database.

From a personal perspective, this project unified the core competencies developed across the MSAI program: prompt engineering, schema validation, AI system architecture, full-stack development, and responsible AI deployment. The Veterans Crisis Line integration and the careful prompt guardrails against providing legal or medical advice reflect a commitment to building AI systems that are not only capable but genuinely safe for vulnerable users.

The foundation built in v3 positions NextMission Navigator for future expansion including persistent user accounts, database-backed saved plans, expanded geographic resource coverage, and deeper ElevenLabs voice interaction. The project represents both a technically complete capstone deliverable and the first production release of a platform with real potential for veteran community impact.

Deliverables Summary

Component	Details
Live Website	www.VetNavi.AI – publicly accessible, fully functional
Source Code	github.com/Msmetamorphosis (private – potential IP)
AI Action Plan Generator	Goal classification + RAG resource injection + Anthropic Claude + Zod validation
AI Chatbot	Multi-turn text chat with voice input and Veterans Crisis Line integration
Voice AI Agent	ElevenLabs ConvAI web component embedded on all pages
Resource Directory	20+ verified resources across 6 categories with regional filtering
Final Report	This document
Video Presentation	Narrated demo available via course submission

Appendix A: Project Plan & Gantt Chart

The full project plan and Gantt chart were submitted as separate documents (Indy-1-NextMissionNavigator-ProjectPlan.pdf and Indy-1-NextMissionNavigator-Ganttchart-Estimate.pdf). Key milestones are summarized in the Version Control section above. Total estimated development effort: 200 hours across the semester.

Appendix B: Screen Mockups

The live application is accessible at www.VetNavi.AI. The capstone project landing page, including all navigation links to project documents, is hosted at https://msmetamorphosis.github.io/nextmission-capstone/

Appendix C: Architecture Diagrams

Architecture diagrams were included in the Software Design Document (Indy-1-NextMission_Navigator-Design.pdf). The diagrams illustrate: (1) High-level subsystem architecture showing Frontend, Backend API, Intelligence Engine, Knowledge Curation Layer, and Voice Module interactions; (2) System Architecture Diagnostic Matrix comparing subsystem components; (3) Experimental Design overview; and (4) Behavioral Robustness Funnel showing data flow through the RAG pipeline.

The core data flow can be summarized: User Input → Frontend (React) → /api/action-plan (Next.js Route) → goalClassify.ts → resourceMatch.ts → generateActionPlan.ts → Anthropic Claude API → actionPlanOutput.ts (Zod validation) → attachCatalogNotes() → ActionPlanResponse → Frontend Render.