Team Experience66

Technological Feasibility
October 17, 2026

Team Members:
Ethan Meyer, Manjot Kaur, Moraa Morara, Nasya Valenzuela

Client:
Mike Talyor, Mark Manone

Mentor:
Scott LaRocca

Overview:

The purpose of this document is to evaluate the technical feasibility of developing the Route 66 Companion app, which connects travelers to historical and cultural stories along Route 66 using location-based digital content. It outlines key challenges such as mapping, data integration, and offline access, and explains how tools like Kotlin, Mapbox, Node.js, and SQLite will be used to create a stable, responsive, and offline-capable system. Overall, this phase confirms that the project is technically achievable and lays a strong foundation for building an engaging, interactive experience that brings Route 66's history to life for travelers.

Table of Contents

Introduction

Route 66, also known as the “Mother Road,” is more than a highway — it’s a living record of America’s history, culture, and identity. As it nears its 100th anniversary, millions of travelers, historians, and enthusiasts are getting ready to celebrate its legacy. From quirky roadside stops to the deep histories of Indigenous communities, Route 66 holds countless stories worth preserving and sharing. Yet in today’s digital world, much of this heritage stays hidden in archives, out of reach for the people traveling its path.

Problem

Access to Route 66’s historical materials is limited to static systems like CONTENTdm. These platforms are strong in data storage but not designed for travel use or mobile discovery. Cline Library’s Special Collections and Archives (SCA) at Northern Arizona University holds over 100,000 digitized items — photos, oral histories, maps, and manuscripts — but travelers have to know exactly what to search for and where. This gap between place and content means many visitors miss out on the stories behind what they see. The current system lacks features like location awareness, mobile access, and interactive storytelling. Library staff, while skilled and dedicated, are limited by the tools available and the need for manual curation.

Goal

To close this gap, we are creating a mobile app called Route 66 Companion that brings historical content to users based on their location. Whether traveling the road or exploring from home, users can view photos, hear oral histories, and explore materials linked to specific places.

Key features include:

- Geofenced content based on GPS
- Interactive maps and photo galleries
- Audio playback of oral histories
- Admin dashboard with data and visual reports
- Optional navigation to nearby attractions and upcoming sites

Impact

The Route 66 Companion turns archives into an interactive travel experience. By connecting history with geography, it helps users discover and connect with the stories of the road in a personal way. For Cline Library, it expands public reach, boosts engagement with collections, and offers insights from user data. For travelers, it transforms a simple drive into a guided trip through time. This project is not just about making an app — it’s about keeping Route 66’s stories alive and accessible for everyone, everywhere. The road is waiting, and so are the stories. Now lets look into the Technological Challenges we may face, possible solutions as well as the integration of these solutions.

1.)Technological Challenges

- **Android Development Environment**

  We need a stable and flexible Android development setup to ensure smooth integration of mapping, geofencing, and offline caching. Choosing between Kotlin, Java, or hybrid frameworks (like React Native) will determine our app's long-term performance and maintainability.

- **Mapping Platform and Location Awareness**

  We must detect a user's current or entered location and surface nearby Route 66 resources in real time. This includes designing geofences around points of interest to trigger content dynamically and maintaining responsiveness even in motion.

- **CONTENTdm API Integration**

  The app must query and stream data from Cline Library's CONTENTdm collection, which contains over 100,000 digitized items. We need to handle large datasets, missing metadata, and inconsistent formats while keeping response times low for mobile users.

- **Offline Access and Network Resilience**

  Route 66 passes through areas with little or no connectivity. The system must support caching of maps, metadata, and thumbnails for offline use and synchronize data when users reconnect. This ensures the app remains functional even without internet access.

- **Audio Playback and Transcripts Integration**

  The app must support seamless playback of oral histories tied to locations, with readable transcripts and recovery options for poor connections. Audio streaming should integrate smoothly with our caching model for offline use.

- **Directions and Route Suggestions**
  Users should be able to get directions to attractions and see what's coming up next along the route. This requires a navigation hook and an efficient ordering strategy for nearby points of interest (POIs).

- **Admin Dashboard and Analytics**

  We will build a secure, role-gated dashboard for administrators to view analytics on app usage and user engagement. This requires careful handling of telemetry, privacy, and role-based access controls.

- **Metadata Quality and Provenance**

  Since CONTENTdm records may have inconsistent data, we need a way to normalize and standardize metadata for reliable display in the app. This challenge directly connects to the middleware data-cleaning step in our architecture.

- **Technology Integration and System Architecture**

  Bringing all technologies together into a single system requires careful integration of front-end, middleware, and data components. This includes synchronizing map SDKs, API calls, caching, and analytics pipelines.

  Having identified these core technical challenges, we now turn to a detailed analysis of the technologies available to meet them, beginning with the foundation of the app — its Android development environment.

2.)Technology Analysis

Before diving into specific technologies, it's important to connect these challenges to the tools that can solve them. Each issue, from mapping accuracy to offline caching, depends on selecting the right frameworks, languages, and SDKs that can handle real-world Route 66 conditions. The following analysis evaluates our available options, testing results, and reasoning behind each major technical decision. Together, these choices form the foundation of our app's overall feasibility and performance.

a.) Android Development Environment

Issue Introduction:

The foundation of the Route 66 Companion app is an Android client. Because the app relies on location awareness, offline access, and mapping integrations, choosing the right development environment and programming language directly affects performance, maintainability, and scalability. Our early decision centered around whether to use Kotlin or JavaScript (through a hybrid framework) and which development environment best supports our mapping SDKs.

Desired Characteristics:

An ideal Android development setup should be:

- Fully compatible with native Android features such as GPS and background services.

- Supported by a robust development environment with strong debugging and emulator tools.
- Lightweight and efficient, minimizing dependency overhead.
- Open-source friendly and well-documented for long-term maintainability.

b.) Alternatives:

1. **Kotlin with Android Studio (Native)** – Kotlin is the modern, official language for Android development. It offers concise syntax, full interoperability with Java, and strong integration with Android Studio. Kotlin supports Jetpack Compose for modern UI design and provides better safety through nullability handling.

2. **JavaScript with React Native (Hybrid)** – Allows cross-platform development using JavaScript. However, integrating native Android SDKs like Mapbox or handling offline map caching is less direct and may require additional native bridges.

3. **Java (Legacy Native)** – While stable and mature, Java lacks some of the modern language features and tooling improvements found in Kotlin. Most new Android documentation and libraries now favor Kotlin.

c.) Analysis:

After testing initial builds by loading a basic world map in the Android Studio emulator, Kotlin provided the smoothest integration with both Mapbox SDK and Android Studio's emulator tools. It allowed easy dependency management through Gradle and streamlined use of Android's location services. React Native, while flexible, introduced complexity when integrating offline map features. Java remained an option but lacked modern syntax advantages.

Criteria	**Kotlin (Native)**	**JavaScript (Hybrid)**	**Java (Legacy)**
Performance	5	3	4
Integration Ease	5	3	4
Maintenance	5	3	4
Library Support	5	4	4

Offline Capability	5	3	4

(Scale: 1 = poor, 5 = excellent)

d.) Chosen Approach:

We selected Kotlin with Android Studio as our primary development stack. This environment provides direct access to native Android APIs, streamlined SDK integration, and superior performance for mapping and location-based tasks. Kotlin's interoperability ensures that our project remains maintainable and adaptable as it grows.

e.) Proving Feasibility:

Our first prototype will be developed in Android Studio using Kotlin, demonstrating live GPS tracking and geofence triggering through simulated Route 66 waypoints. This will confirm the effectiveness of the Android environment and Kotlin's stability for core app functions.

2. Mapping Platform and Location Awareness

a.) Issue Introduction:

The app's foundation is an interactive map that reacts to the user's real-world location. It must detect GPS coordinates, display Route 66 attractions, and trigger related content when travelers approach specific areas. This mapping logic drives the user experience and determines the system's scalability and responsiveness. We need a solution that can handle custom styling, geofencing, routing, and integration with Android while remaining developer-friendly.

b.) Desired Characteristics:

An ideal mapping SDK should be:

- Accurate and reliable for real-time GPS tracking.
- Customizable for thematic map styles tied to Route 66's historical aesthetic.
- Lightweight and performant on Android devices.
- Offline-capable to ensure the app works in low-signal areas.
- Cost-effective given our project's academic and public nature.

These criteria matter because the app's main interaction happens through the map. If location updates lag or geofences trigger late, users will lose trust in the experience.

c.) Alternatives:
We compared two major SDKs: ArcGIS Runtime SDK for Android and Mapbox Maps SDK for Android.

- **ArcGIS Runtime SDK (Esri):** Enterprise-grade GIS toolkit developed by Esri, the same ecosystem used by MetroPlan and the NAU GES department. It supports advanced cartography, 3D layers, and integration with ArcGIS Online. However, it has a steeper learning curve, heavier dependencies, and higher licensing costs.

- **Mapbox SDK:** Open-source friendly mapping framework known for its modern styling tools, fast rendering, and easy Android integration. It supports custom tiles, offline maps, and dynamic location tracking. Mapbox has a growing community of developers and usage-based pricing that fits our academic prototype.

d.) Analysis:
We ran sample projects in both SDKs to evaluate setup complexity, map responsiveness, and styling control. ArcGIS required more configuration and a dedicated ArcGIS account setup for each developer. Mapbox's Kotlin integration worked directly within Android Studio with fewer dependencies. Map rendering and geolocation performance were noticeably smoother in Mapbox, and offline region downloads were simpler to implement.

Criteria	**ArcGIS Runtime**	**Mapbox SDK**
Setup/Integration	3	5
GPS Responsiveness	4	5
Custom Styling	3	5
Offline Mapping	4	5
Cost	2	4
Developer Support	5	5

e.) Chosen Approach:

Mapbox SDK is our chosen mapping platform. Its ease of integration, offline support, and pricing make it a strong fit for our goals. We plan to build the initial prototype using Mapbox's location and geofencing APIs, while keeping ArcGIS in mind if the project later requires enterprise-level data hosting.

f.) Proving Feasibility:

We will develop a Mapbox-based prototype that simulates user travel along Route 66 using test GPS data. The app will display markers and trigger content cards for nearby landmarks, demonstrating real-time geofencing accuracy and offline functionality.

3. CONTENTdm API Integration

a.) Issue Introduction:

The Cline Library's CONTENTdm system hosts over 100,000 digitized photos, maps, and oral histories related to Route 66. Our app must pull these items dynamically and display them as users approach related sites. The challenge lies in handling large datasets efficiently, managing missing metadata, and ensuring consistent image loading without overwhelming the mobile client.

b.) Desired Characteristics:

The integration solution should:

- Support fast and reliable API queries with pagination.
- Handle image caching to reduce repeated requests.
- Be resilient to missing or malformed records.
- Allow metadata normalization for consistent display.
- Maintain low network usage to support mobile users.

c.) Alternatives:

- **Direct API Integration:** Query the CONTENTdm REST API directly from the app. Simple but limited error handling and heavier mobile-side processing.
- **Middleware Service (Node.js backend):** Build a lightweight server between CONTENTdm and the app to handle caching, normalization, and filtering before

sending data to users.

- **Static Preloaded Dataset:** Preload key records for each Route 66 stop, updated manually or periodically. Reduces flexibility but ensures offline reliability.

d.) Analysis:

We compared these approaches by testing sample queries and load times. Direct API calls worked but produced inconsistent JSON structures. Middleware caching through Node.js allowed us to standardize fields (title, description, coordinates) and compress image metadata before sending it to the mobile client. Preloading static data was the fastest but limited the app's ability to show newly added content.

Criteria	Direct API	Middleware Service	Preloaded Data
Response Speed	3	4	5
Data Accuracy	3	5	4
Maintenance Effort	4	3	2
Flexibility	5	5	2
Offline Readiness	2	3	5

e.) Chosen Approach:

We selected the Middleware Service approach. It provides a balance between speed, accuracy, and flexibility. Our plan is to build a Node.js/Express backend that queries CONTENTdm, caches frequent requests, and delivers simplified JSON data to the app. This minimizes mobile processing and prepares us for analytics tracking later.

f.) Proving Feasibility:

Our prototype will include a Node.js demo endpoint that retrieves sample Route 66 photo data filtered by coordinates. We'll connect this to the Android client to confirm that data transfers correctly, proving that our backend pipeline supports large-scale CONTENTdm integration.

4. Offline Access and Network Resilience

a.) Issue Introduction:

Route 66 spans long stretches of rural terrain where users may lose service for hours. To maintain continuity, the app must continue showing maps, audio, and photos offline. Designing for offline use is technically challenging, as it requires smart caching and fallback behaviors without inflating storage usage.

b.) Desired Characteristics:

- Local caching of maps, thumbnails, and metadata.
- Partial downloads that only fetch nearby regions.
- Graceful degradation when the network is unavailable.
- Syncing mechanisms to update data when back online.
- Low power and storage consumption.

c.) Alternatives:

- **Mapbox Offline Regions:** Built-in functionality allowing predefined areas to be downloaded for offline use.
- **Custom SQLite Cache:** Store key metadata and image URLs locally using Room or SQLite.
- **Hybrid Cache Strategy:** Combine Mapbox's offline maps with a local SQLite cache for metadata and thumbnails.

d.) Analysis:

Testing showed that Mapbox's built-in offline regions worked well for map tiles but didn't store custom data like CONTENTdm results. SQLite handled metadata caching efficiently but couldn't manage map tiles on its own. The hybrid model offered the best of both worlds, fast offline map display with flexible metadata access.

Criteria	**Mapbox Offline**	**SQLite Cache**	**Hybrid Model**
Map Availability	5	1	5
Metadata Access	2	5	5
Implementation Complexity	4	3	4

Storage Efficiency	3	5	4
User Experience	4	3	5

e.) Chosen Approach:

We will adopt the Hybrid Cache Model, using Mapbox's offline regions for maps and a small SQLite layer for metadata and thumbnails. This ensures continuity in low-signal areas without overloading devices.

f.) Proving Feasibility:

We'll demonstrate this through an offline simulation in Android Studio's emulator, disabling mobile data and confirming that cached maps and media remain available. This prototype will prove that hybrid caching maintains usability even with complete network loss.

Technology Integration

Now that we have looked at the individual challenges, like mapping, geofencing, CONTENTdm integration, caching, audio playback, routing, and analytics, the next step is to bring everything together into a single unified system. THis section shows how all the pieces connect to form the overall architecture for the Route 66 Companion app. Its essentially the first big picture view of how data flows, what connects to what, and how users will experience the app in real time.

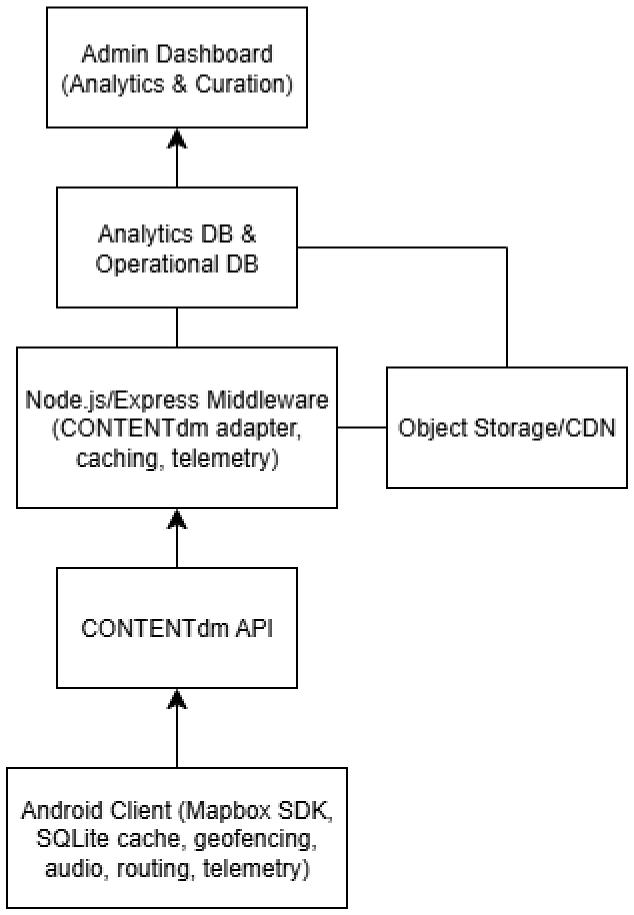
The app's foundation is the **Android mobile client.** This is where travelers will see the map, get location-based content, and interact with the audio, images, and transcripts. It uses the Mapbox SDK to track GPS, handle geofences, display landmarks, and keep the map working offline. A small SQLite cache stores important POI metadata and thumbnails, so that the app still works even when the signal is weak or drops out completely.

Behind the scenes is the **Node.js/Express middleware**. This layer connects the app to CONTENTdm, cleans and normalizes metadata, caches frequent requests, and sends lightweight JSON back to the client. It also collects telemetry data (what POIs are viewed and when), making sure it's handled in a privacy-safe way. This approach reduces the work the mobile device has to do and keeps things fast on the road.

For media, larger assets like audio and images will be stored in object storage/CDN. The app only loads what it needs when it needs it, which helps performance. For navigation, there will be a routing hook that lets users deep-link into their preferred map app for turn-by-turn directions.

On the library side, an **admin dashboard** connects to the analytics layer and operational database. This gives curators the ability to review usage patterns, fix metadata issues, and improve content over time; it's secure and role-gated, so only authorized users can make changes or view analytics.

System Diagram


Admin Dashboard (Analytics & Curation)
Analytics DB & Operational DB
Node.js/Express Middleware (CONTENTdm adapter, caching, telemetry)
Object Storage/CDN
CONTENTdm API
Android Client (Mapbox SDK, SQLite cache, geofencing, audio, routing, telemetry)

How the parts work together:

1. **Location triggers content.** As the user drives and enters a geofence, the app calls the middleware, which fetches content from CONTENTdm and returns it in a clean, ready-to-display format.
2. **Offline support.** When there's no signal, the app falls back on cached maps and thumbnails so the experience continues smoothly.
3. **Media delivery.** Large files like images and audio are pulled from storage/CND for faster performance and less strain on the middleware.
4. **Routing.** A simple hook connects users to navigation apps for upcoming sites.
5. **Analytics and curation.** The middleware logs events and pushes them to the analytics layer, where the dashboard helps staff improve content and track engagement.

This architecture connects all the smaller solutions into one system that meets the project goals, its location-aware, offline-friendly, performance-focused, and supports long-term content management and improvement.

Conclusion

Finding the best way to get historical and relevant information to travelers is our top priority. Our goal is to build a Route 66 road companion that highlights landmarks, shares oral histories, and creates a meaningful on-the-road experience. From library archives to a user friendly road companion the team has had to work through different challenges. Throughout this document, we addressed the main challenges and how we worked through them. By combining Mapbox, CONTENTdm, caching, audio playback, routing, and analytics into one system, we've set a clear technical foundation. Using Mapbox for mapping, CONTENTdm for digital collections, a Node.js middleware for handling data, SQLite caching for offline use, and an admin dashboard for analytics gives us a clear flexible technical path forward. Even though this is the early stage, this plan positions us to design, prototype, and refine a tool that will make Route 66 history more accessible and meaningful for travelers.