Kentuckiana Planning and Development Agency (KIPDA) iOS Application Individual Report

Obadah Abozaid with Team Members:

Clare Kunkel, Chace Washington, Stephen Slaten, Parker Harpool, Keegan Burnett

KIPDA Supervisors:

Jessica Elkin and Beth Mathis

Overview

Throughout the Spring 2025 semester, I played a major role in advancing our iOS application's feature development, focusing particularly on enhancing user interaction capabilities and maintaining an effective source control. My primary technical contribution centered on implementing a sophisticated chat feature that enables direct user-to-user communication within the application. This development required extensive work with Firebase's real-time database and authentication systems, including a significant update to the Firebase SDK to ensure compatibility with current best practices and resolve several deprecated method issues.

A substantial portion of my contribution involved modernizing our development infrastructure and establishing efficient workflows. I took the initiative to restructure our version control system, implementing a more systematic branching strategy that follows the industry’s best practices. This included establishing a clear hierarchy with main as our production code base, a develop branch for integration, and feature branches for ongoing development work. This restructuring required careful attention to existing branches from the previous semester, many of which needed to be rebased or archived to maintain a clean and manageable repository structure.

To facilitate seamless team collaboration, I set up a remote development environment by configuring SSH access to a Mac machine over the University of Louisville's Wi-Fi network. This innovation proved valuable as it enabled team members to compile and test iOS code even from Windows machines - where none of us had Mac OSX - effectively removing a significant barrier to development participation. Further, I maintained the project's GitHub repository, serving as the primary administrator for our remote origin, and managed access controls for team collaborators.

In the realm of cloud infrastructure, I expanded our application's capabilities by implementing Firebase Functions to support the chat feature. This involved careful configuration of the Firebase Console and the addition of IAM profile support to enable Google’s VertexAI integration within the application. These enhancements laid the groundwork for more sophisticated user interaction and AI-driven features in future iterations of the application.

As a team leader, I took on the responsibility of coordinating work among team members, ensuring that our development progress remained both efficient and timely at a pace. This required regular communication with team members, task prioritization, and management of our development timeline. I maintained a strong focus on code quality and accessibility, ensuring that all team members could effectively contribute to the project regardless of their physical location (mostly) or development environment.

The implementation of the chat feature represented a significant technical challenge that required deep understanding of both iOS development principles and Firebase's real-time communication protocols. This feature not only enhanced the application's functionality but also

demonstrated our commitment to creating a more engaging and interactive user experience. The work involved careful consideration of user privacy, data security, and real-time performance optimization.

The chat functionality in our iOS application represents a sophisticated integration of real-time messaging capabilities with AI-powered assistance, using Firebase's real-time database and Google's Vertex AI platform. The implementation consists of several key technical components:

GeminiService Integration:

The application implements a GeminiService class that interfaces with Vertex AI through the Firebase Vertex AI SDK. This service is initialized with the "gemini-2.0-flash-001" model, specifically configured for KIPDA's use case. The service handles both chat session management and response generation, implementing error handling through custom enumerated error types (GeminiServiceError). Key technical features include:

```
1. class GeminiService {
2.     private var model: GenerativeModel
3.     private var chat: Chat?
4.
5.     init() throws {
6.         model = VertexAI.vertexAI().generativeModel(modelName: "gemini-2.0-flash-
001")
7.         chat = model.startChat()
8.     }
9. }
10.
```

Chat Architecture:

The chat system is built on a MVVM (Model-View-ViewModel) architecture, with three primary components:

Data Models:

ChatMessage: A comprehensive struct that handles different message types (text, AI, system) with Codable conformance for Firebase integration

Conversation: Manages conversation metadata including participants, timestamps, and message history

ViewModel Layer (ChatViewModel):

Implements real-time synchronization with Firebase

Manages conversation state and message handling

Coordinates between the UI and the GeminiService

Handles user authentication state and permissions

View Layer:

Implements a responsive UI with real-time message updates

Features a dual-mode interface supporting both AI and user-to-user chat

Includes typing indicators and message status updates

Implements message rendering through LazyVStack

Firebase Integration:

The application uses Firebase Firestore for real-time data synchronization, implementing key features:

Real-time Message Synchronization:

```
1. func loadMessages(for conversationId: String) {
2.     messagesListener = db.collection("conversations")
3.         .document(conversationId)
4.         .collection("messages")
5.         .order(by: "timestamp")
6.         .addSnapshotListener { [weak self] snapshot, error in
7.             // Real-time message updates handling
8.         }
9. }
10.
```

Conversation Management:

Implements conversation creation and management

Handles participant tracking and message history

Manages unread message counts and last message tracking

Security Rules:

Implements proper authentication checks

Ensures message privacy through participant-based access control

Manages user permissions and conversation access

AI Integration Features:

The chat system implements several sophisticated AI-related features:

Context Management:

Maintains conversation context for more relevant responses

Implements proper error handling for AI service disruptions

Manages conversation state between AI and user interactions

Response Generation:

```
func generateResponse(to message: String) async throws -> String {
    let responseStream = try chat.sendMessageStream(message)
    var fullResponse = ""

    for try await chunk in responseStream {
        if let text = chunk.text {
            fullResponse += text
        }
    }
    return fullResponse
}

```

Specialized AI Behavior:

Implements KIPDA-specific knowledge base integration

Handles domain-specific queries about senior services

Maintains consistent AI personality across conversations

User Experience Optimizations:

Message Handling:

Implements efficient message queuing and delivery

Manages message state (pending, delivered, error)

Handles message retry logic for failed deliveries

Performance Optimizations:

Implements message pagination for large conversations

Uses lazy loading for message history

Optimizes image and media handling

UI Responsiveness:

Implements smooth scrolling and animation

Manages keyboard interactions and view adjustments

Handles device rotation and different screen sizes

This technical implementation demonstrates a sophisticated integration of modern iOS development practices, cloud services, and AI capabilities, creating a robust and scalable chat system that serves both user-to-user and AI-assisted communication needs.

The ChatView implementation represents my integration of real-time messaging capabilities with AI-assisted communication in our iOS application. At its core, the view utilizes SwiftUI's state management system to handle dynamic user interactions and real-time updates. The implementation begins with a state management structure:

```
1. struct ChatView: View {
2.     @EnvironmentObject var authManager: AuthenticationManager
3.     @StateObject private var chatViewModel = ChatViewModel()
4.     @State private var messageText = ""
5.     @State private var selectedChatType: ChatType = .ai
6.     @State private var showUserSelector = false
7.     @State private var scrollToBottom = false
8. }
9.
```

The view architecture is built around a hierarchical component structure, with each element serving a specific purpose in the user interaction flow. The header section incorporates the KIPDA logo and contextual information about the current chat session, changing its display based on whether the user is interacting with the AI assistant or another user. This adaptive interface is done through conditional rendering:

```
 1. HStack {
 2.     Image("kipda_logo")
 3.         .resizable()
 4.         .scaledToFit()
 5.         .frame(height: 60)
 6.
 7.     Spacer()
 8.
 9.     if selectedChatType == .user && chatViewModel.currentConversation != nil {
10.         Text(chatViewModel.currentUserName)
11.             .font(.headline)
12.     }
```

```
13. }
14.
```

One of the key innovations in my implementation is the dual-mode chat system, which allows users to seamlessly switch between AI-assisted communication and user-to-user messaging on the same tab. This functionality is done through a custom picker interface that manages the transition between modes:

```
 1. Picker("Chat Type", selection: $selectedChatType) {
 2.     Text("AI Assistant").tag(ChatType.ai)
 3.     Text("User Chat").tag(ChatType.user)
 4. }
 5. .onChange(of: selectedChatType) { newValue in
 6.     if newValue == .ai {
 7.         chatViewModel.messages = []
 8.         Task {
 9.             await chatViewModel.startAIChat()
10.         }
11.     } else if newValue == .user {
12.         chatViewModel.messages = []
13.         showUserSelector = true
14.     }
15. }
16.
```

The message display system employs SwiftUI's LazyVStack for efficient rendering of chat messages, particularly important when handling lengthy conversations. This implementation includes an automatic scrolling mechanism that ensures the most recent messages are always visible:

```
 1. ScrollViewReader { scrollView in
 2.     ScrollView {
 3.         LazyVStack(spacing: 12) {
 4.             ForEach(chatViewModel.messages) { message in
 5.                 MessageBubble(message: message)
 6.                     .id(message.id)
 7.             }
 8.             Color.clear
 9.                 .frame(height: 1)
10.                 .id("bottomID")
11.         }
12.     }
13.     .onChange(of: chatViewModel.messages.count) { _ in
14.         withAnimation {
15.             scrollView.scrollTo("bottomID", anchor: .bottom)
16.         }
17.     }
18. }
19.
```

The MessageBubble component represents a significant advancement in the message display system. It implements a design that differentiates between user messages, AI responses, and system notifications through visual styling and positioning:

```
struct MessageBubble: View {
    let message: ChatMessage
    private let currentUserId = Auth.auth().currentUser?.uid

    private var isCurrentUser: Bool {
        message.senderId == currentUserId
    }

    private var isAI: Bool {
        message.senderId == "AI"
    }
}

```

The entire system is designed with lifecycle management in mind, ensuring a proper initialization of conversations and cleanup of orphaned resources. This includes handling view appearance and disappearance, managing chat sessions, and maintaining proper state synchronization throughout the application:

```
.onAppear {
    chatViewModel.loadConversations()
    Task {
        if selectedChatType == .ai {
            await chatViewModel.startAIChat()
        }
    }
}

```

This comprehensive implementation creates a fully functional, user-friendly chat interface that successfully integrates both AI-assisted and user-to-user communications. The system maintains performance even with numerous messages, is built to handle network conditions gracefully, and provides immediate feedback for user actions. The modular design allows for easy maintenance and future enhancements, while careful attention to user experience details ensures that the chat functionality remains intuitive and accessible to all users – especially the elderly.

Self-Development

Throughout this capstone project, I experienced significant professional growth and technical skill development, in areas that were previously unfamiliar to me. While my

background included experience with .NET MAUI and iOS platform development, this project presented new challenges that required substantial learning and adaptation.

The transition to native iOS development using SwiftUI and Xcode was very different from my previous experience. Unlike MAUI's cross-platform approach, working directly with SwiftUI required understanding Apple's native frameworks and design patterns. This transition involved learning SwiftUI's declarative syntax, some state management systems, and iOS-specific lifecycle management. The learning curve was steep when implementing features like custom view modifiers, handling navigation stacks, and managing view hierarchies in a purely SwiftUI environment.

Another major area of growth was in the implementation of advanced AI capabilities through Google's Vertex AI platform. While I had previous experience with Firebase services, integrating a large language model (LLM) presented new challenges. This required an understanding of some introductory prompt engineering, managing API rate limits for KIPDA, and implementing efficient response handling. The implementation of the GeminiService class was particularly tricky and I ran into many issues implementing it. This implementation required learning about asynchronous programming patterns in Swift, handling streaming responses, and managing conversation context across multiple interactions.

Source control management brought unique challenges in our project setup. While I had previous experience with version control systems, managing a shared repository with multiple contributors working from a single local machine required learning new workflows. This included implementing proper branching strategies, managing merge conflicts with consideration for multiple users, and maintaining clean commit histories. The experience taught me valuable lessons about collaborative development practices and the importance of clear communication in version control management for archival purposes and future work.

The project also pushed me to develop better error handling and debugging skills, particularly in the context of real-time applications. Implementing the chat feature required careful consideration of edge cases and potential failure points. The project also required learning about Firebase's more advanced features, such as implementing real-time data synchronization and managing complex data relationships. This included understanding Firebase's security rules, optimizing database queries, and implementing efficient data listeners.

These technical challenges and learning experiences have significantly enhanced my capabilities as a software developer. The project has not only improved my technical skills but also developed my ability to learn and adapt to new technologies quickly. The experience of working with SwiftUI, Vertex AI, and advanced Firebase features has broadened my understanding of modern application development and prepared me for future challenges in the field.

Trends & Emerging Applications

The implementation of AI-assisted communication and real-time chat features in our KIPDA application aligns with several significant emerging trends in the healthcare and social services sectors. The integration of large language models (LLMs) through Google's Vertex AI platform parallels with a broader industry shift toward more intelligent, context-aware assistance systems. This trend is relevant in healthcare and social services, where organizations are increasingly leveraging AI to improve access to information and support for elderly and vulnerable populations.

The use of LLMs in customer service and information distribution represents a rapidly growing trend across industries. According to recent market analyses by Dell Technologies, the global conversational AI market is expected to grow from $6.8 billion in 2021 to $18.4 billion by 2026, with healthcare and public services being key growth sectors. Our implementation of the GeminiService for KIPDA's specific domain demonstrates how organizations can customize large language models to provide accurate, context-specific information while maintaining a human-like interaction quality. This approach is valuable in the healthcare and social services sector, where access to accurate information about services, benefits, and resources is crucial but often complicated by complex eligibility requirements and changing regulations.

The real-time chat functionality implemented in our application reflects another significant industry trend: the increasing demand for immediate, personalized communication channels between service providers and clients. This feature becomes relevant in the context of aging-in-place initiatives and remote healthcare services, where direct, efficient communication between seniors, caregivers, and service providers is essential. The dual-mode chat system, supporting both AI-assisted and human-to-human communication, represents an emerging hybrid approach that many organizations are adopting to balance automation with a more personal touch.

The integration of Firebase's real-time database for message synchronization aligns with the industry's move toward more responsive, real-time applications. This trend is important where timely communication can be critical. Our implementation demonstrates how modern cloud infrastructure can support secure, HIPAA-compliant communication channels while maintaining high performance and reliability. The security and privacy considerations in our implementation reflect growing industry concerns about data protection, particularly in healthcare-adjacent applications.

Looking forward, the modular design of the chat system positions well for emerging trends in healthcare technology, such as the integration of telehealth services, automated health reminders, and care coordination features. The system's architecture allows for future expansion to include features like automated appointment scheduling, medication reminders, and

integration with electronic health records systems of the administrators' choosing, all of which are growing trends in healthcare technology.

The mobile-first approach of our SwiftUI implementation aligns with the increasing reliance on mobile devices among both healthcare providers and patients. This trend is relevant for organizations like KIPDA that serve diverse populations with varying levels of technical proficiency. The intuitive interface with accessibility built-in – including localization and color correction with dark mode - and responsive nature of our implementation support the industry's move toward more accessible, user-friendly healthcare technology solutions.

These implementations and architectural decisions position our application at the crossroads of several key industry trends: the rise in AI healthcare services, the growing importance of real-time communication in service delivery, and the increasing focus on secure, accessible mobile solutions for healthcare and social services. As these trends continue to evolve, our application's modular design and use of modern technologies provide a solid foundation for future enhancements and adaptations as needed.

The integration of AI assistance in healthcare applications is also part of a broader trend toward expanding access to healthcare information and services. By providing an AI-powered interface that can answer questions about senior services, benefits, and resources, we're contributing to the industry's movement toward more accessible and equitable healthcare information systems. This is important for organizations like KIPDA that serve populations who may face barriers to accessing traditional healthcare information channels.

Further, the implementation of real-time chat capabilities reflects the healthcare industry's recognition of the importance of immediate, accessible communication channels. As healthcare services become increasingly digital, the ability to provide both automated and human support through a single platform is becoming a crucial feature for healthcare organizations. This hybrid approach allows organizations to efficiently handle routine inquiries through AI while maintaining the option for human intervention when more complex or sensitive issues arise.

Conclusions

The KIPDA iOS mobile application represents a significant advancement in how social service organizations can utilize modern technology to serve their communities more effectively. Through the implementation of features such as real-time chat with AI assistance, comprehensive service information access, event management, and secure user authentication, we have created a platform that not only meets current needs but is positioned to evolve with future requirements. The application bridges the gap between traditional social services and modern digital accessibility, providing KIPDA's clientele with a user-friendly interface to access critical information and support.

The technical achievements in this project, particularly my own integration of Google's Vertex AI through the GeminiService, the implementation of real-time Firebase synchronization, and the development of a native iOS interface using SwiftUI, demonstrate the potential for innovative solutions in the social services sector. The application's architecture emphasizes security, scalability, and user experience, making it a valuable tool for both service providers and clients. As KIPDA continues to serve the aging population and their caregivers in the Kentucky-Indiana region, this application proves how thoughtful technology implementation can enhance the delivery of social services. The project not only meets its immediate goals of improving service accessibility and communication but also establishes a foundation for future enhancements and adaptations as the needs of the community and the capabilities of technology continue to evolve.