CSS 545 A - Mobile Computing

HW3 - State Management

Professor Hansel Ong
Raviteja Tanikella

Project Name: ATSPro

Various App States In ATSPro

The various states my app can enter, along with how it behaves in each of these state and why I must consider them is as follows:

1. **Switch Between Home and Results Page?**
    - When the user enters the app and provides the job description, uploads the resume and clicks on “Generate”, the app directs to the results page where there are 4 tabs (Overview, Missing Keywords, Suggestions, and Interview Prep) and the respective responses from the Gemini model API.
    - The user returns to the home page by clicking on home, and only can enter the results page again when he clicks on generate. As soon as he clicks on generate, the previous chat history and responses from the Gemini model will be cleared and the new results will be displayed. This is crucial to avoid conflicts while searching for different jobs.

2. **When the user leaves the app (Not Exit)?**
    - When the user leaves the app and returns to the app again, he’ll be joined where he left and his responses and chat conversation is restored and he can continue his conversation seamlesly.
    - This ensures the user's conversation is restored so that when he returns back to the app he can continue from where he left, which should be the case with every app for a seamless and hassle free usage.

3. **When the user exits the app?**
    - When the user exits the app, all his search results generated by Gemini model and his conversation is entirely cleared because the user exited the app. Upon returning to the app, the app starts fresh from the Home where the user can provide details and generate responses and start a new conversation with the Gemini model.
    - This mechanism ensures that all the conversation data and responses are cleared upon exiting the app as it means restarting the app again. (Maybe this conversation history can be restored for the user upon integrating cloud storage in the future scope).

Tombstone (Suspend/Resume) Management in ATSPro

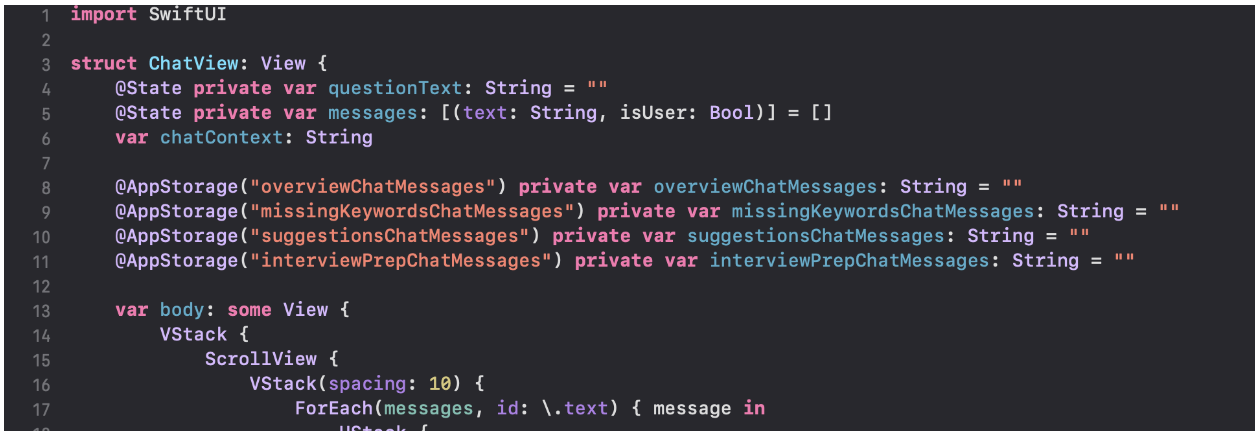
Tombstoning refers to the process of preserving the state of an application when it is temporarily suspended or put in the background, and restoring that state when the app resumes. This behavior is crucial for a good user experience because it allows users to seamlessly pick up where they left off, even after being away from the app for a while.

User Going Away and Coming Back to the Same State:
In our specific application, this behavior is particularly important because users might want to switch between tabs in the app or temporarily leave the app without losing their chat history. Here's how the app achieves this behavior effectively:

1. **Maintaining Separate Chat Messages for Each Tab:**
   - The app has four distinct chat tabs (Overview, Missing Keywords, Suggestions, and Interview Prep).
   - Each tab has its own separate chat history that needs to be preserved independently.
   - This is managed using the @AppStorage property wrapper in SwiftUI, which provides a convenient way to read and write values that are automatically persisted across app launches.

2. **Navigating Away and Returning to the Same State:**
   - When the user navigates away from the app (e.g., to another app or to the home screen), the chat history for each tab is preserved using @AppStorage.
   - When the user returns to the app, the chat history is restored for the respective tab they were interacting with, allowing them to continue their conversation seamlessly.

3. **Implementation Using @AppStorage:**
   - The ChatView SwiftUI view utilizes @AppStorage to store and retrieve chat messages.
   - For example, each tab's chat history is stored under a unique key (e.g., overviewChatMessages for the Overview tab).
   - The loadMessages and saveMessages functions handle loading and saving the chat messages to and from @AppStorage.



```swift
import SwiftUI

struct ChatView: View {
    @State private var questionText: String = ""
    @State private var messages: [(text: String, isUser: Bool)] = []
    var chatContext: String

    @AppStorage("overviewChatMessages") private var overviewChatMessages: String = ""
    @AppStorage("missingKeywordsChatMessages") private var missingKeywordsChatMessages: String = ""
    @AppStorage("suggestionsChatMessages") private var suggestionsChatMessages: String = ""
    @AppStorage("interviewPrepChatMessages") private var interviewPrepChatMessages: String = ""

    var body: some View {
        VStack {
            ScrollView {
                VStack(spacing: 10) {
                    ForEach(messages, id: \.text) { message in
```

- The loadMessages function reads the saved messages for the respective tab and populates the chat accordingly, while the saveMessages function writes the current chat state to @AppStorage.

4. **Seamless User Experience:**
    - The implementation ensures that users have a seamless experience when navigating away from the app and coming back.
    - By preserving the chat state, the app creates a consistent and predictable environment for the user, enhancing their overall experience.