# TABLE OF CONTENTS

[Gem Voice Assistant — Project Report 1](#_Toc864332518)

[1. Real-World Problem Identification 1](#_Toc327751764)

[2. Proposed Solution 1](#_Toc716170447)

[Key Features: 2](#_Toc246834245)

[3. Responsive User Interfaces 2](#_Toc1593411132)

[Key Design Principles: 2](#_Toc2109354963)

[MOBILE VIEW: 3](#_Toc855012040)

[IPAD VIEW: 4](#_Toc93237233)

[LAPTOP VIEW: 6](#_Toc1767539050)

[4. Data Storage 6](#_Toc1476353807)

[Why have I used Shared-Preferences? 7](#_Toc1716662863)

[5. APIs / Packages / Plug-ins Used 7](#_Toc6007140)

[6. Issues and Bugs Encountered and Resolved during Development 8](#_Toc425904966)

[References 10](#_Toc1834815476)

# Gem Voice Assistant — Project Report

## **1. Real-World Problem Identification**

In today’s digital world, people rely heavily on virtual assistants for information retrieval, daily tasks, and automation. However, most existing assistants like Google Assistant or Siri are limited to specific platforms or devices and do not allow full customization or integration with other apps.

Many users, especially students and professionals, require a **lightweight, customizable voice assistant** that:

* Responds intelligently to natural voice commands.
* Can provide AI-generated responses to general queries.
* Works across Android, iOS, and web platforms.
* Saves past responses for quick review.

Hence, the problem identified is the **lack of a flexible, AI-integrated, cross-platform voice assistant** that can understand speech, generate responses intelligently, and provide both text and audio feedback within a Flutter-based environment.

## **2. Proposed Solution**

To address this problem, we developed **Gem Voice Assistant**, a Flutter-based AI voice assistant app that:

* Listens to the user’s speech via **Speech-to-Text (STT)**.
* Sends the recognized text to **Gemini AI** for processing and intelligent response generation.
* Speaks out the Gemini response using **Text-to-Speech (TTS)**.
* Saves and displays previous interactions using **local storage**.
* Supports a **responsive interface** across mobile, tablet, and desktop.

### **Key Features:**

1. ️**Speech Recognition:** Converts spoken commands into text using **speech-to-text** package.
2. **AI Response Generation:** Uses Google’s **Gemini API** (**google generative-ai)** to generate intelligent, contextual responses.
3. **Voice Feedback:** Speaks out AI responses using flutter\_tts.
4. **Persistent Storage:** Stores chat history locally via **shared preferences.**
5. **Responsive UI:** Interface adapts automatically for various screen sizes.

This solution provides both **intelligence** and **accessibility,** acting as a smart, responsive, and portable AI companion.

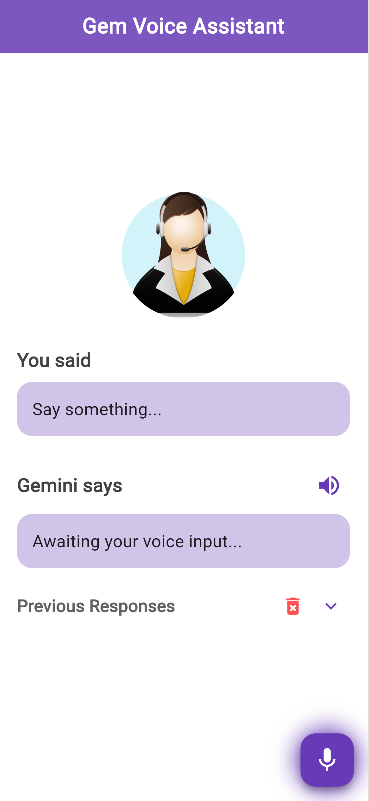
## **3. Responsive User Interfaces**

The UI of Gem Voice Assistant was designed using Flutter’s responsive layout system to ensure it adapts perfectly on all devices.  
 The layout centers the main content on larger screens while keeping an easy-to-use structure on smaller mobile devices.

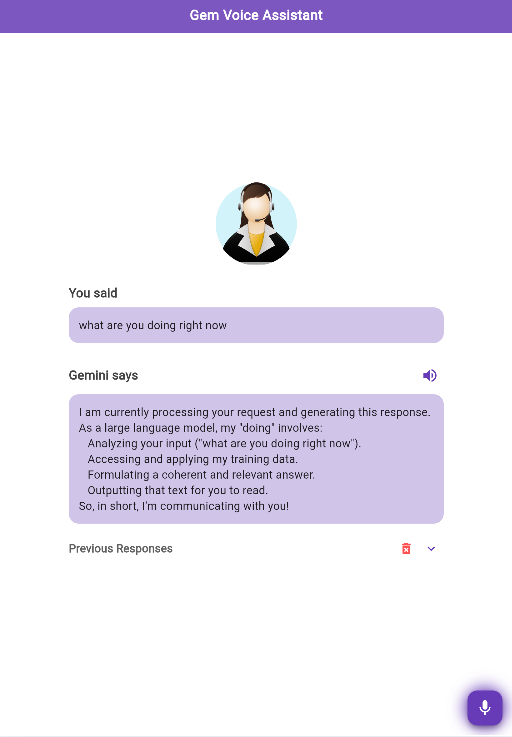
### **Key Design Principles:**

* **Center alignment with constraints:**  
   The app uses ConstrainedBox(maxWidth: 600) inside Center() to keep a consistent look on all devices.
* **Material Design:**  
   Uses Flutter’s Material widgets and color theming for a professional appearance.
* **Dynamic feedback:**  
   Displays animations like CircularProgressIndicator during AI processing.

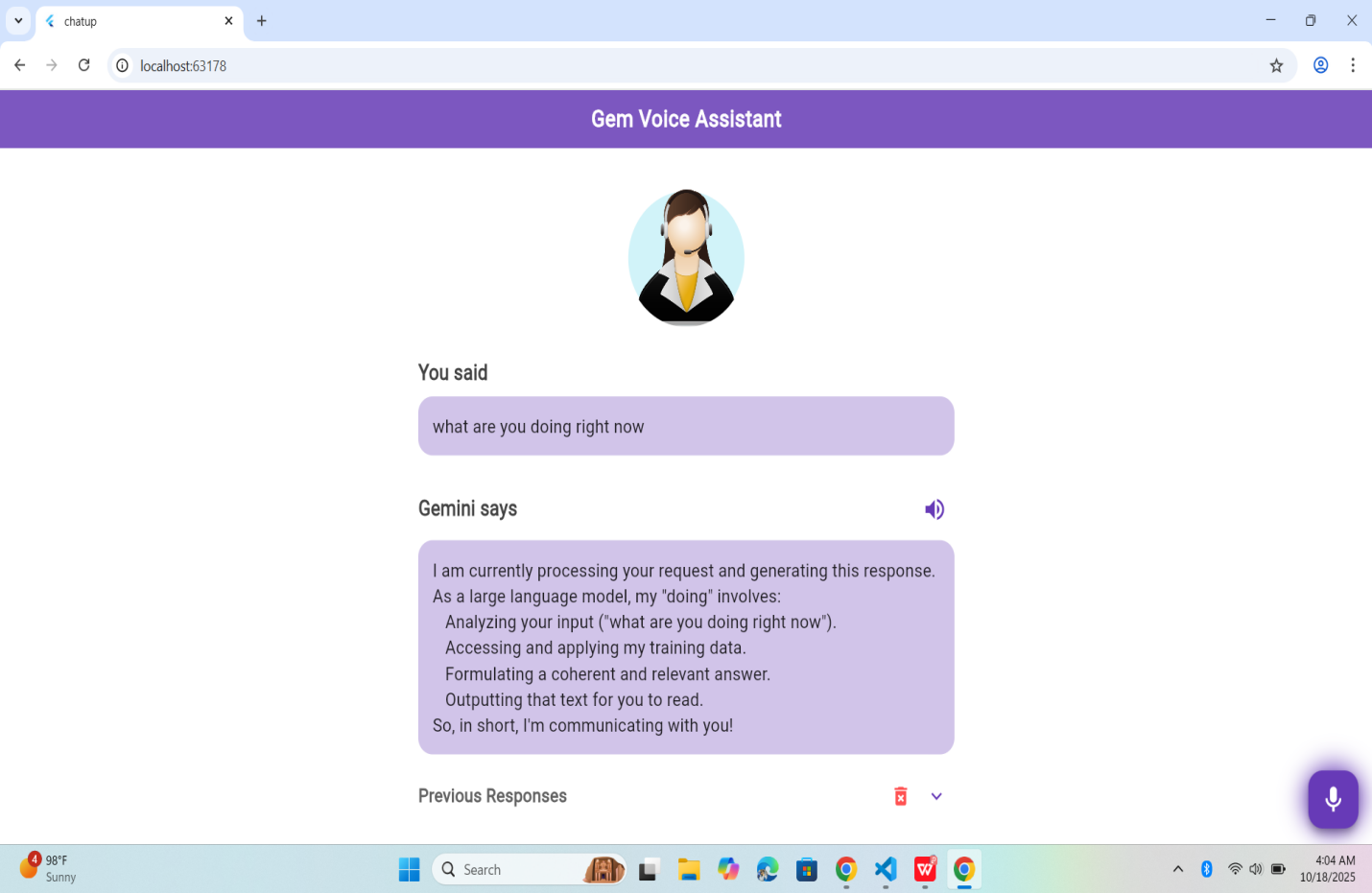
# MOBILE VIEW:



# IPAD VIEW:



# LAPTOP VIEW:



## **4. Data Storage**

The application uses Shared-Preferences for data storage.

### **Why have I used Shared-Preferences?**

* It allows storing simple data such as strings, Booleans, and lists persistently.
* Ideal for lightweight local storage like saving previous Gemini responses.
* It doesn’t require setup of an external database, making it perfect for mobile and Flutter apps.
* Faster read/write operations compared to file-based or SQL-based systems for small datasets.

**Example Use:**



This approach ensures that **response history remains available** even when the app is closed and reopened.

## **5. APIs / Packages / Plug-ins Used**

|  |  |  |
| --- | --- | --- |
| **Package / API** | **Purpose** | **Justification** |
| **GoogleGenerativeAi** | To connect with Gemini API for AI-generated responses. | Provides direct access to Google’s Gemini model with simple integration. |
| **Speech-to-text** | Converts spoken input to text. | Enables real-time voice recognition  with minimal latency. |
| **Flutter.tts** | Converts AI text responses back into spoken audio. | Provides natural voice output for user-friendly experience. |
| **sharedPreferences** | Stores AI responses locally. | Lightweight and perfect for saving short text data persistently. |
| **http** | Used for network calls (if needed in API configuration). | Ensures smooth communication with Gemini API or other services. |

# 6. Issues and Bugs Encountered and Resolved during Development

1. A security-related issue was found in my project; the API key was originally written directly in the main code file. To prevent exposure of credentials, the key was moved to a separate secrets.dart file, which is excluded from version control.
2. This is the issue that got me confused for long, while testing the Gemini API, a “**Model not found” error** appeared because the model's name used in the code was outdated. Updating it from 'gemini-1.5-flash' to 'gemini-2.5-flash' fixed the issue.
3. On larger screens like tablets or desktops, the layout was stretching too much and looked unbalanced. This was fixed by wrapping the main content inside a Center() widget and using a ConstrainedBox with a maximum width of 600, which keeps the layout responsive and visually centered on all screen sizes.
4. Another problem occurred when the Gemini API returned no text, causing the app to crash due to a null value. This was solved by adding a null check in the code using response.text ?? "No response from Gemini.", which ensures that even if the API doesn’t return text, a safe fallback message is shown.
5. The Gemini API also returned responses with unwanted Markdown symbols such as \*, #, or \_, making the text look cluttered. This issue was solved by implementing a custom cleaning function called cleanGeminiResponse(), which uses regular expressions to remove unnecessary symbols before displaying the text to the user.
6. Another issue appeared with overlapping voice outputs. When the assistant generated new responses, the previous audio would still play. This happened because Flutter TTS does not automatically stop a previous speech before starting a new one. To fix this, the line await flutterTts.stop(); was added before starting new speech with flutterTts.speak(text).

## **References**

* [Flutter Official Documentation](https://flutter.dev/)
* [Google Generative AI API Docs](https://developers.generativeai.google)
* [speech\_to\_text Package](https://pub.dev/packages/speech_to_text)
* [flutter\_tts Package](https://pub.dev/packages/flutter_tts)
* [shared\_preferences Package](https://pub.dev/packages/shared_preferences)