INDUSTRIAL ORIENTED MINI PROJECT

Report

On

ESSENTIAL VOICE DELIVERY SYSTEM -BRIDGING THE GAP BETWEEN UNLETTERED

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

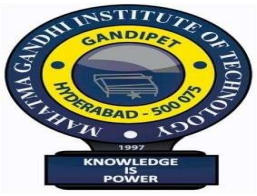
Battula Satyanarayana Reddy - 22261A1209

Thota Advith - 22261A1260

Under the guidance of

Mrs.B Meenakshi

Associate Professor, Department of IT



DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY
(AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), RangaReddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **ESSENTIAL VOICE DELIVERY SYATEM – BRIDGING THE GAP BETWEEN UNLETTERED** submitted by **Battula Satyanarayana Reddy (22261A1209), Thota Advith (22261A1260)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Mrs. B Meenakshi**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:

Mrs. B Meenakshi
Associate Professor
Dept. of IT

IOMP Supervisor:

Dr. U. Chaitanya
Assistant Professor
Dept. of IT

EXTERNAL EXAMINAR

Dr. D. Vijaya Lakshmi
Professor and HOD
Dept. of IT

DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **ESSENTIAL VOICE DELIVERY SYSTEM – BRIDGING THE GAP BETWEEN UNLETTERED** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Mrs . B Meenakshi , Associate Professor**, Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

Battula Satyanarayana Reddy - 22261A1209

Thota Advith - 22261A1260

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our Internal supervisor **Mrs.B Meenakshi, Associate Professor**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our IOMP supervisor **Dr. U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi** Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

Battula Satyanarayana Reddy - 22261A1209

Thota Advith -22261A1260

ABSTRACT

Access to essential goods is critical for ensuring the well-being of individuals in all communities. However, many unlettered individuals—those who can speak but cannot read or write—face significant barriers when accessing ration services, which traditionally rely on written communication. This project aims to develop an AI-powered Voice-Based Ration Ordering System that enables users to place orders for daily necessities using simple voice calls in their native languages.

By leveraging multilingual speech-to-text technology and Natural Language Processing (NLP), the system processes verbal inputs to extract key order details such as item names, quantities, and delivery preferences. It then confirms the interpreted order with the user before finalizing and forwarding it for processing and doorstep delivery. The system is designed to function effectively across multiple regional languages, ensuring inclusivity and ease of use for people with varying linguistic backgrounds.

Users can interact with the system through a basic mobile phone, without the need for internet access or reading skills. This voice-driven approach removes the need for intermediaries, empowers users to order independently, and improves access to essential goods in underserved areas.

With this voice-based solution, the project seeks to bridge the digital and literacy divide, promote digital empowerment, and ensure that no one is left behind in the delivery of basic services.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 MOTIVATION

Many individuals, especially in rural and underserved areas, are unable to read or write, even though they can communicate verbally. Most ration and essential goods ordering systems rely on written forms or mobile apps, making it difficult for unlettered people to access services independently.

Depending on others for placing basic orders can lead to inconvenience, delays, and even misuse. So, it is necessary to integrate voice technology with public service delivery. With the help of speech-to-text technology and Natural Language Processing (NLP), we can develop a system that understands voice commands and places orders accurately.

The model will help the users to speak their needs naturally, confirm the order, and receive their essentials hassle-free, without the need for reading or writing.

1.2 PROBLEM STATEMENT

In many low-literacy and rural communities, accessing essential goods like ration items is a major challenge due to inability to read or write. Traditional ration ordering systems rely heavily on written forms or mobile apps, which are not user-friendly for unlettered individuals. This often leads to dependency on others, errors, or missed deliveries. The proposed system solves this by using voice technology and speech-to-text processing to let users place orders through simple phone calls in their own language.

Additionally, language remains a major barrier in accessing government services. Most systems operate in English or a few regional languages, making it hard for users from diverse linguistic backgrounds. To overcome this, the system supports multilingual voice input and confirmation in the user's mother tongue, ensuring that the process is inclusive, clear, and accessible for everyone—regardless of literacy level.

1.3 EXISTING SYSTEM

Various technologies have been developed to support voice interaction and accessibility for low-literate or illiterate users, particularly in rural settings. **Speech-to-text systems** utilize natural language processing (NLP) and automatic speech recognition (ASR) engines to convert spoken language into text in real-time (Bano et al., 2020). While they support multiple languages, their accuracy drops in noisy environments, they often require internet access, and they are not specifically designed for ordering essential goods like ration items.

Voice user interfaces (VUIs) enable illiterate users to interact with systems using voice commands and basic keypad inputs (Restyandito & Chan, 2014). These systems are mostly seen in academic or experimental setups and typically lack advanced NLP features, making them unable to process complex requests or provide confirmation and feedback to users.

Mobile interfaces for low-literacy users focus on intuitive, icon-based or voice-assisted designs that simplify interactions for novice users (Medhi et al., 2011). While user-friendly, these interfaces still require basic interaction skills like tapping icons and are often not connected to backend systems, limiting their ability to complete tasks like order placement and tracking.

Multilingual speech recognition systems allow users to communicate in their native languages using multilingual recognition models and language modeling techniques (Saloni & Singh, 2020). However, these systems often perform poorly with regional dialects and are primarily used in commercial platforms such as Google Assistant or Alexa. They also lack the ability to extract structured data, such as specific order items.

Finally, **UI design for illiterate and semi-literate users** is an emerging area with research focused on best practices using user-centric frameworks and audio-visual guidance (Islam et al., 2023). While these guidelines are useful, they remain largely theoretical and are not available as complete platforms. They require further integration with voice and ordering technologies to deliver end-to-end services suitable for real-world applications.

1.3.1 Limitations of Existing Systems

- **Limited Contextual Adaptability**: Current voice systems are generic and not specifically designed for structured tasks like ration ordering. They fail to accurately extract item names, quantities, or delivery preferences from informal speech patterns.
- **Dependence on Internet Connectivity**: Most speech-to-text systems and voice assistants require constant internet access, making them impractical in low-network or offline rural environments.
- **Lack of End-to-End Functionality**: Existing voice interfaces often stop at converting speech to text and do not complete the entire order lifecycle, including confirmation, processing, and delivery.
- **Insufficient Regional Language Support**: Many systems support only major languages and do not perform well with local dialects or rural accents, limiting their usability in linguistically diverse areas.
- **Absence of Confirmation Feedback Loop**: Voice assistants do not usually read back or confirm the extracted order details, increasing the risk of miscommunication or incorrect deliveries.
- **Not Integrated with Ration Supply Chains**: Current systems are not connected to backend ration shop vendors or delivery systems, making them unsuitable for real-world deployment in public distribution systems.

1.4 PROPOSED SYSTEM

The proposed system is a voice-based AI solution designed to help unlettered users place ration orders easily using a regular phone call. When a user speaks their order, the system uses speech-to-text technology to convert the spoken words into written text. Then, using Natural Language Processing (NLP), it extracts key details like item names, quantities, and delivery preferences.

The system is unique because it supports multiple languages, allowing users to speak in their own mother tongue. Once the order is understood, the system reads back the details

to the user for confirmation. After the user confirms, the order is forwarded for processing and delivered to their doorstep.

This system removes the need for reading, writing, or using complex apps—making it simple, accessible, and inclusive for unlettered individuals, especially in rural areas.

1.4.1 Advantages

- **Voice-Based Ordering Accessibility**: The system empowers unlettered users to place orders using voice commands, eliminating the need for reading or writing and making it highly suitable for rural and low-literacy populations.
- **Multilingual Voice Support**: By supporting multiple regional languages, the system ensures effective communication with users from diverse linguistic backgrounds, enhancing inclusivity and user comfort.
- **Confirmation Before Processing**: A unique voice feedback mechanism reads back the interpreted order to the user for confirmation, ensuring accuracy and reducing the chances of order mismatch or misunderstanding.
- **Independence from Internet and Smartphone**: The system is designed to work even with basic phones and minimal internet, making it accessible to users in low-connectivity or technologically underserved areas.
- **Scalable and Modular Design**: The architecture allows easy integration of more languages, item types, and delivery methods, making it adaptable for other public services beyond ration distribution.
- **User Empowerment and Transparency**: The voice interaction loop enables users to independently manage their ration orders without third-party dependence, fostering dignity, autonomy, and trust in public service systems.

1.5 OBJECTIVES

- To develop a voice-based system that allows users to place ration orders through simple phone calls without needing to read or write.
- To implement speech-to-text conversion for processing voice inputs in multiple regional languages.
- To use Natural Language Processing (NLP) to extract key order details such as item names, quantities, and delivery preferences.
- To provide voice-based confirmation in the user's mother tongue before finalizing the order.
- To ensure the system is inclusive, accessible, and easy-to-use for unlettered individuals in rural and underserved areas.

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

Software Requirements

- Speech Recognition Libraries like Google Speech-to-Text, Vosk, or Mozilla DeepSpeech
- Natural Language Processing (NLP) tools like spaCy or NLTK
- Python libraries such as Flask, NumPy, and requests
- Front-end tools (for admin dashboard) like HTML, CSS, and JavaScript
- Backend framework like Flask or Django to handle user input and process orders
- IDE environment like Google Colab or VS Code for development and testing
- **Hardware Requirements**
- Intel i5 or higher processor for smooth system performance
- RAM: 8GB or more to handle voice and language processing efficiently
- Storage: At least 512GB SSD for storing logs, user data, and system files

2. LITERATURE SURVEY

Medhi et al. [1] conducted an ethnographic study combined with controlled experiments to design mobile interfaces for novice and low-literacy users. Their findings showed that graphical and voice-based interfaces performed significantly better than text-based ones, with live human operators offering the highest accuracy. However, the reliance on human intervention raised scalability and cost concerns. They suggested future work focus on developing hybrid UI models that combine speech and visuals, aiming for cost-effective alternatives to live operators in low-resource settings.

Bano et al. [2] developed a prototype speech-to-text (STT) system for multilingual applications, particularly targeting Indian languages. Their system improved inclusivity and demonstrated the feasibility of converting spoken language to text across various regional tongues. Despite promising outcomes, the study highlighted limitations in handling dialectal variation, speaker dependency, and background noise. They recommended advancing STT models with better real-time accuracy and robustness across diverse linguistic and acoustic conditions.

Islam et al. [3] presented a systematic review of 45 studies spanning 2001 to 2022 to explore UI design for illiterate and semi-literate users. They proposed 16 core design principles and five key rules focused on text usage, interaction simplicity, audio-visual cues, pictography, and information architecture. While comprehensive, the study identified a lack of real-world, deployable UI solutions. The authors emphasized the need for context-aware systems, public interface design, and new usability frameworks that bridge research insights with practical implementation.

Restyandito and Chan [4] reviewed existing literature on voice user interfaces (VUIs) with a focus on cognitive usability for illiterate populations. They found that while VUIs enhance accessibility, users often face challenges due to the abstract nature of voice interaction, memory load, and lack of visual anchors. The review stressed the importance of culturally adaptive designs that account for the transient nature of audio and cognitive constraints, suggesting more inclusive and intuitive VUI frameworks.

Saloni and Singh [5] conducted a technical review of automatic speech recognition (ASR) systems, focusing on end-to-end (E2E) architectures for multilingual speech-to-text conversion. Their findings showed that E2E models hold promise for real-time, multi-language applications. However, challenges remain in supporting low-resource languages, managing background noise, and ensuring flexible deployment across different platforms. They proposed future research to address these gaps and improve deployment feasibility in underserved regions.

Author(s)	**Year**	**Title**	**Methodology**	**Findings**	**Research Gaps / Future Work**
Indrani Medhi et al.	2011	Designing Mobile Interfaces for Novice and Low-Literacy Users	Ethnographic study + controlled experiments with mobile UIs	Graphical and voice interfaces performed better than text; live operator most accurate	Explore scalable hybrid designs (e.g., graphical + speech); cost-effective alternatives to human operators
Shahana Bano et al.	2020	Speech to Text Translation Enabling Multilingualism	Prototype STT system tested across Indian languages	Enabled multilingual speech translation; improved inclusivity	Enhance real-time accuracy, robustness across dialects, and speaker independence
Muhammad Nazrul Islam et al.	2023	Designing User Interfaces for Illiterate and Semi-Literate Users: A Systematic Review	Systematic review of 45 articles from 2001–2022	Proposed 16 design principles and 5 UI rules (text, interaction, AV, pictography, info architecture)	Research needed on public UI design, context-aware systems, and new usability frameworks
Restyandito & Alan Chan	2014	A Review on Voice User Interface for Illiterate People	Literature review on cognitive usability and VUI	VUI helps access but users struggle due to abstract thinking and memory load	VUI must consider cognitive constraints, transient nature of audio, and cultural context
Saloni & Dr. Williamjeet Singh	2020	Multilingual Speech to Text Conversion – A Review	Technical review of ASR technologies and architectures	E2E ASR systems are promising; useful for multilingual real-time conversion	Need better support for low-resource languages, noise handling, and flexible deployment

3. ANALYSIS AND DESIGN

The suggested voice-based ration ordering system addresses a critical accessibility gap by enabling unlettered individuals to independently place essential goods orders using natural speech. The system captures voice input via phone call and processes it through advanced Speech-to-Text and Natural Language Processing (NLP) techniques to accurately extract items, quantities, and delivery preferences. The structured order is then confirmed through Text-to-Speech (TTS) feedback, ensuring correctness before submission.

What distinguishes this system is its multilingual voice interface and offline accessibility. The use of regional languages allows users to interact comfortably in their mother tongue, while the voice confirmation mechanism ensures that the user has full control over their order. The final order is processed and a confirmation is sent via SMS or automated voice call, making the system usable even in areas with low internet penetration. The seamless integration of voice technology with public delivery mechanisms ensures that users receive timely, accurate, and personalized service—bridging both the literacy and digital divide.

3.1 MODULES

The modules of this project are:

1. Voice Input Module

- Allows users to place orders through a simple voice call.
- Supports multilingual speech input based on user preference.
- Ensures accessibility for unlettered and digitally unskilled individuals.

2. Speech-to-Text Conversion Module

- Converts spoken input into text using multilingual speech recognition.
- Handles different regional accents and dialects.
- Prepares input for further processing by NLP engine.

3. Natural Language Processing (NLP) Module

- Extracts relevant information from converted text:
    - Item names
    - Quantity
    - Delivery address or preferences

Handles variations in speech and informal language structure

4. Order Management Module

- Structures the extracted information into a valid order format.
- Handles inventory checking and price calculations (if applicable).
- Prepares the order for confirmation.

5. Voice Confirmation & Text-to-Speech (TTS) Module

- Converts the order summary back to voice for user confirmation.
- Asks user to confirm, modify, or cancel the order via voice.
- Ensures user validation before order placement.

6. Delivery & Notification Module

- Sends finalized orders to delivery partners or ration shop vendors.
- Notifies the user of order status via call or SMS.
- Can also collect voice-based feedback post delivery.

3.2 ARCHITECTURE

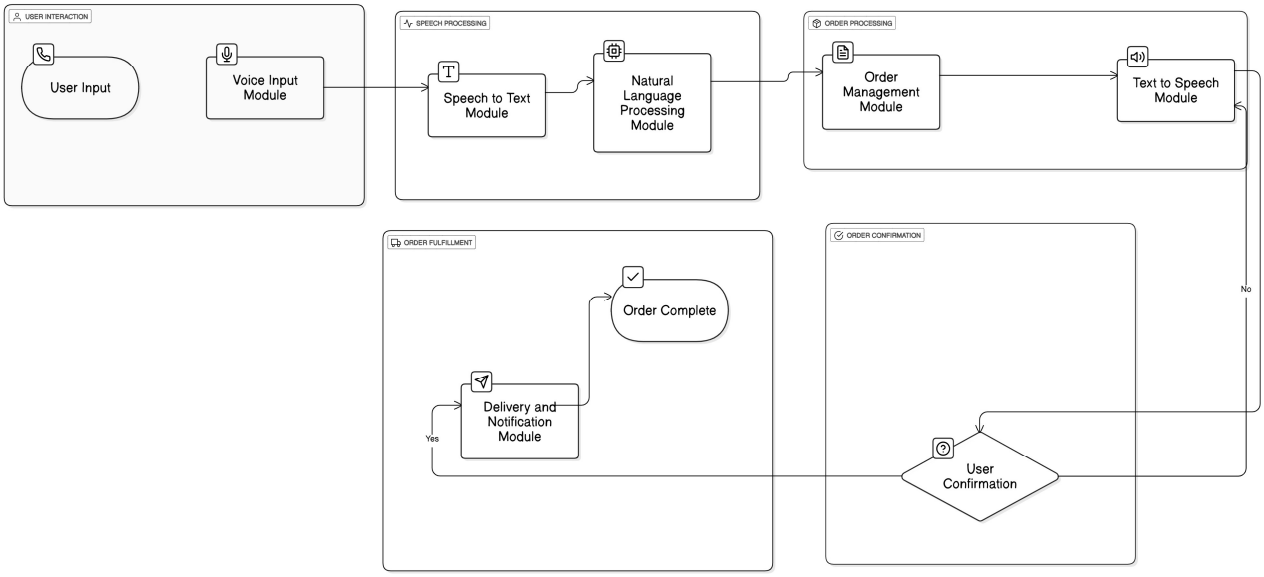


Fig. 3.2.1 Architecture of VOICE-BASED RATION ORDERING SYSTEM

3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAM

A use case diagram is a visual representation that depicts the interactions between various actors and a system, capturing the ways in which users or external entities interact with the system to achieve specific goals. It is an essential tool in system analysis and design, often used in software engineering and business analysis. In a use case diagram, actors are entities external to the system that interact with it, and use cases are specific functionalities or features provided by the system as seen in Fig. 3.3.1.1. These interactions are represented by lines connecting actors to use cases. The diagram helps to illustrate the scope and functionality of a system, providing a high-level view of how users or external entities will interact with it.

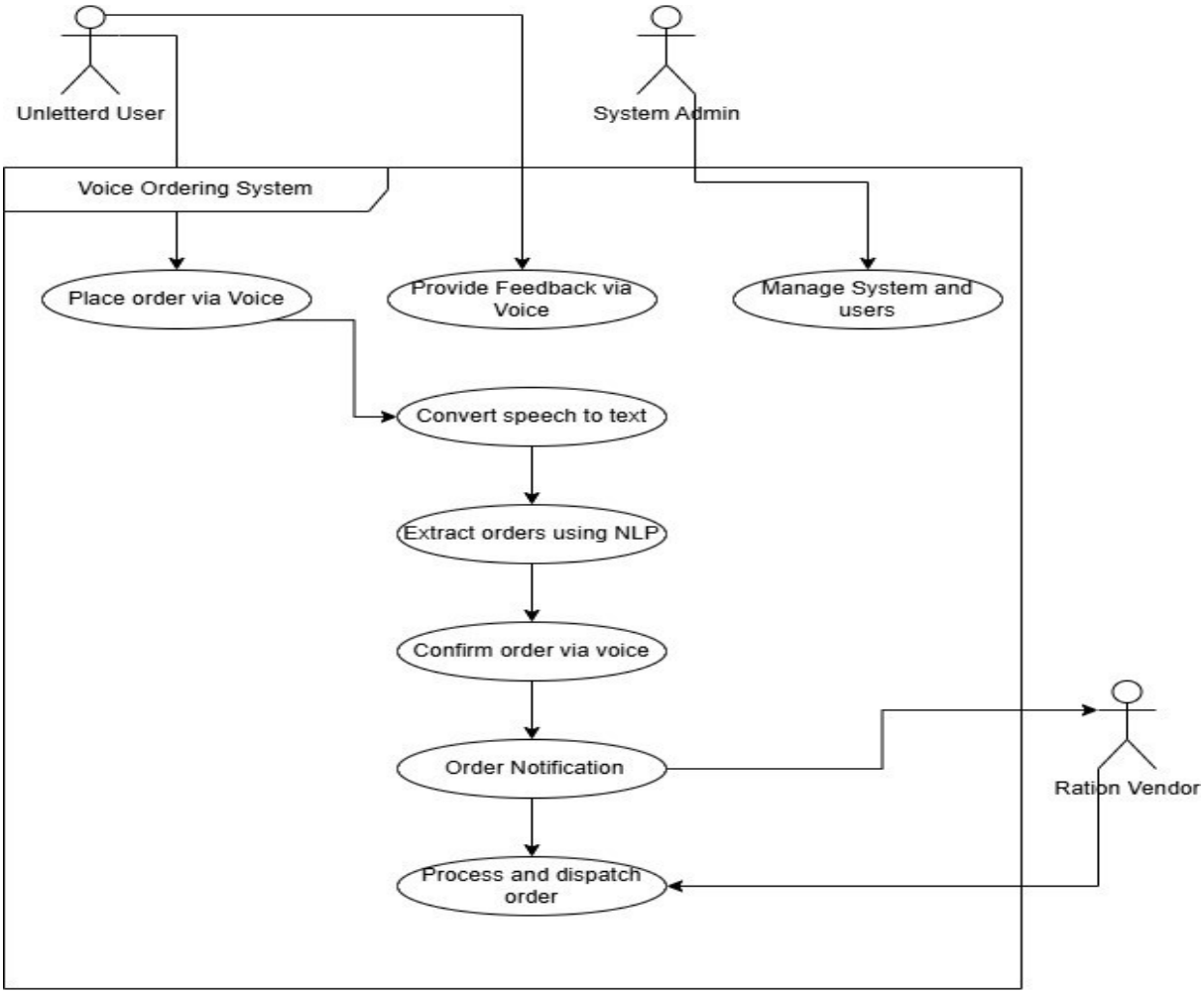


Fig. 3.2.1 Use Case diagram

Actors:

1. User (Unlettered Individual):

   Represents the end user who interacts with the system using their voice to place a ration order via a simple phone call.

2. System:

   The backend application responsible for converting speech to text, extracting order details, confirming orders, and managing delivery and notification.

Use Cases:

1. Give Voice Order:
   The user initiates a phone call and speaks their ration order in their preferred regional language.

2. Convert Speech to Text:
   The system uses a speech recognition engine to convert the spoken input into text.

3. Process and Extract Order:
   The system uses Natural Language Processing (NLP) to identify items, quantities, and delivery preferences from the converted text.

4. Structure the Order:
   A formal, structured order is created based on the extracted information.

5. Voice-Based Confirmation:
   The system reads back the interpreted order using Text-to-Speech (TTS) and prompts the user to confirm or modify it.

6. Submit Order:
   Upon receiving confirmation from the user, the system finalizes and stores the order for delivery processing.

7. Send SMS/Call Notification:
   The system sends a confirmation message via SMS or voice call to the user, ensuring that the information is accessible even without internet connectivity.

3.3.2 CLASS DIAGRAM

A class diagram is a visual representation that models the static structure of a system, showcasing the system's classes, their attributes, methods (operations), and the relationships between them as seen in Fig. 3.4.2.1. It is a key tool in object-oriented design and is commonly used in software engineering to define the blueprint of a system.

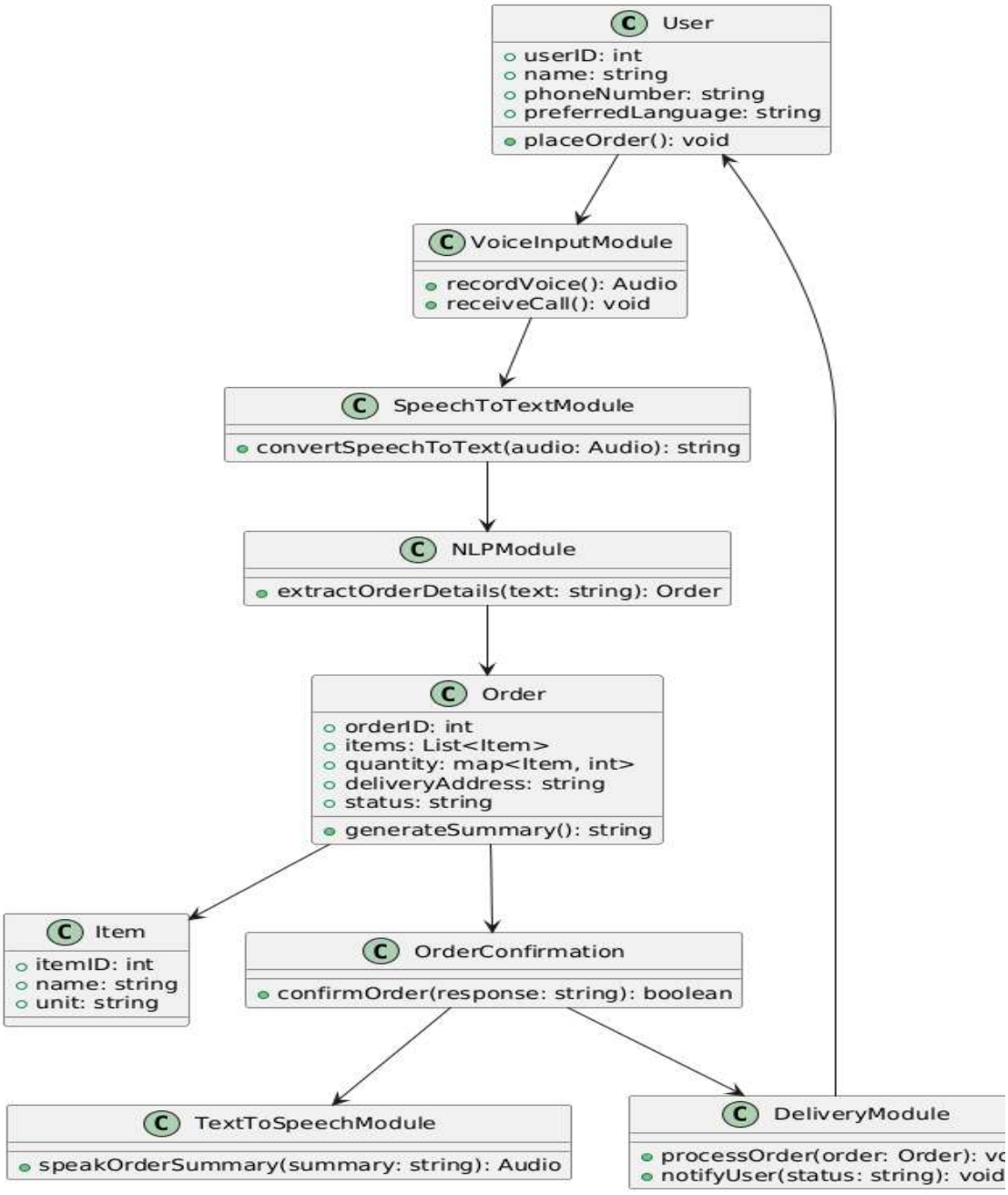


Fig. 3.3.2 Class Diagram

Relationships

1. User → VoiceInputModule:

   The user interacts with the system by making a voice call. The VoiceInputModule captures the audio and initiates the input process.

2. VoiceInputModule → STTModule:
   The captured voice input is passed to the Speech-to-Text (STT) Module, which converts the audio into a textual representation.

3. STTModule → NLPProcessingModule:
   The STT output is forwarded to the NLPProcessingModule, which analyzes the text to extract ration item names, quantities, and delivery details.

4. NLPProcessingModule → OrderManagerModule:
   The extracted structured data is sent to the OrderManagerModule, which constructs a formal order object ready for confirmation.

5. OrderManagerModule → TTSModule:
   The prepared order is passed to the Text-to-Speech (TTS) Module, which converts the order details into voice format for confirmation with the user.

6. TTSModule → User:
   The system reads the order aloud to the user and awaits confirmation (e.g., via a "yes" or "no" response).

7. OrderManagerModule → NotificationModule:
   If confirmed, the order is finalized and forwarded to the NotificationModule, which sends order details to vendors and sends SMS/voice notifications to the user.

System Flow

1. Voice Order Input:
   • The user initiates a call and gives a verbal order.
   • The VoiceInputModule records and prepares the input for processing.

2. Speech-to-Text Conversion:
   • The voice input is converted into text using the STTModule.
   • The text is forwarded to NLP for further extraction.

3. Order Extraction and Structuring:
   • The NLPProcessingModule identifies items, quantities, and delivery info.

• The OrderManagerModule compiles the extracted data into a formal order structure.

4. Voice-Based Order Confirmation:
   • The TTSModule converts the structured order into spoken feedback.
   • The user confirms or corrects the order via voice.

5. Final Order Processing and Notification:
   • Upon confirmation, the order is passed to the NotificationModule.
   • The user receives confirmation via SMS or automated voice call.
   • The vendor receives the order for delivery execution.

3.3.3 ACTIVITY DIAGRAM

An Activity Diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the flow of control or data through the system as seen in Fig. 3.3.3.1. It focuses on the flow of activities and actions, capturing the sequence of steps in a particular process or workflow. Activity diagrams are commonly used to model business processes, workflows, or any sequential activities in a system.

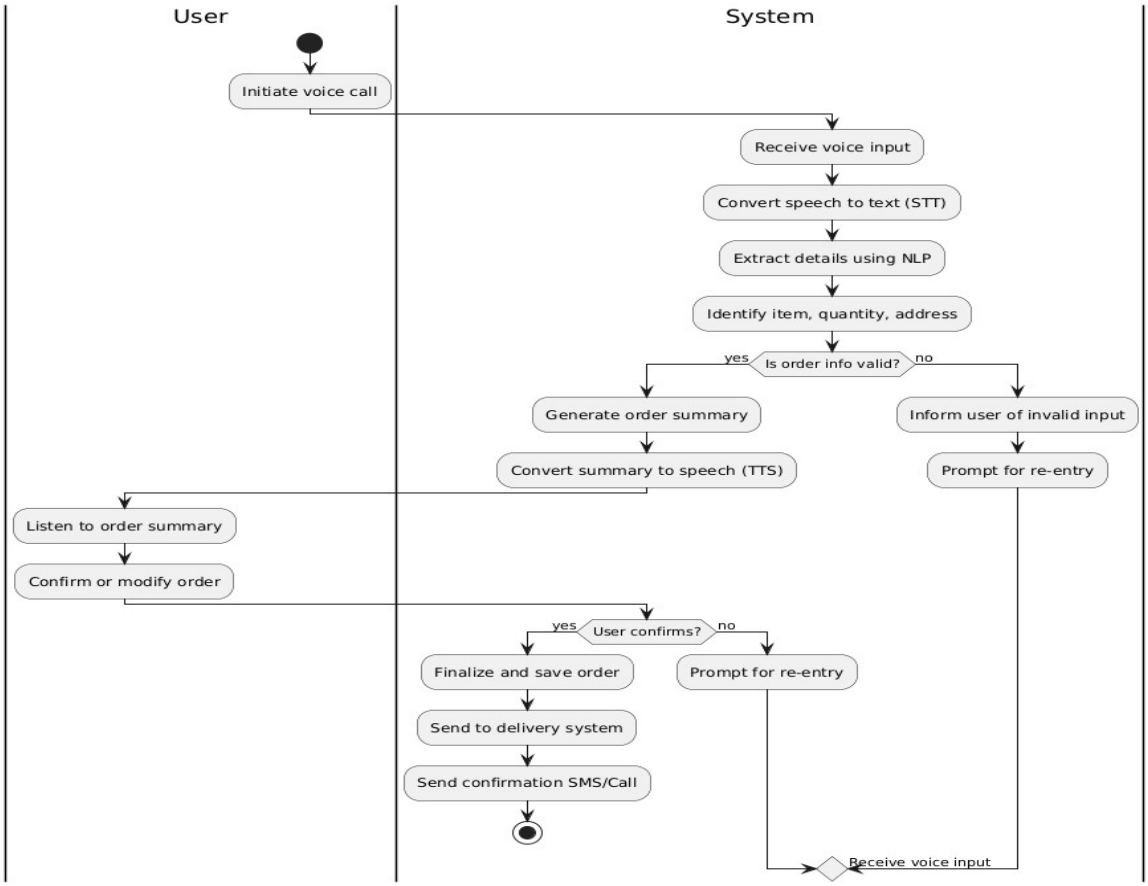


Fig. 3.3.3 Activity Diagram

Flow Explanation

1. Initiate Phone Call

• The user begins by initiating a phone call to the ration ordering system using a basic mobile phone.

• No app or internet access is required.

2. Speak Ration Order

• The system greets the user in their regional language and prompts them to speak their ration order.

• The user says their order naturally (e.g., “2 kg rice, 1 liter oil, deliver tomorrow”).

3. Capture and Convert Voice Input
• The voice is captured by the **Voice Input Module**.
• The **Speech-to-Text (STT) Module** converts the spoken input into text format.

4. Analyze Order Text using NLP
• The **Natural Language Processing (NLP) Module** extracts key elements from the text such as:

- Item names (e.g., rice, oil)
- Quantities (e.g., 2 kg, 1 liter)
- Delivery instructions (e.g., deliver tomorrow)

5. Structure and Prepare Order
• The extracted details are compiled into a formal order structure by the **Order Management Module**.

6. Voice-Based Confirmation
• The **Text-to-Speech (TTS) Module** reads the interpreted order back to the user:
"You have ordered 2 kg rice and 1 liter oil for delivery tomorrow. Say 'yes' to confirm."

7. User Confirmation Decision
• A decision point is reached:

- If the user confirms with "yes", the system proceeds.
- If the user says "no", the system allows them to repeat or cancel the order.

8. Finalize and Store Order
• Once confirmed, the order is saved and forwarded to the delivery backend.

9. Notify Vendor and User
• The order details are sent to the relevant ration vendor or delivery partner.
• The user receives a confirmation via:

- SMS message
- Automated voice call (for low-literacy users)

10. Order Delivery Process Begins
• The vendor processes the order and schedules the delivery.
• The system may also support tracking or follow-up communication if needed.

11. End of Workflow

• The voice session ends.

• The user may initiate a new order by calling again, or simply disconnect.

3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in Fig. 3.3.4.1, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behaviour of a system.

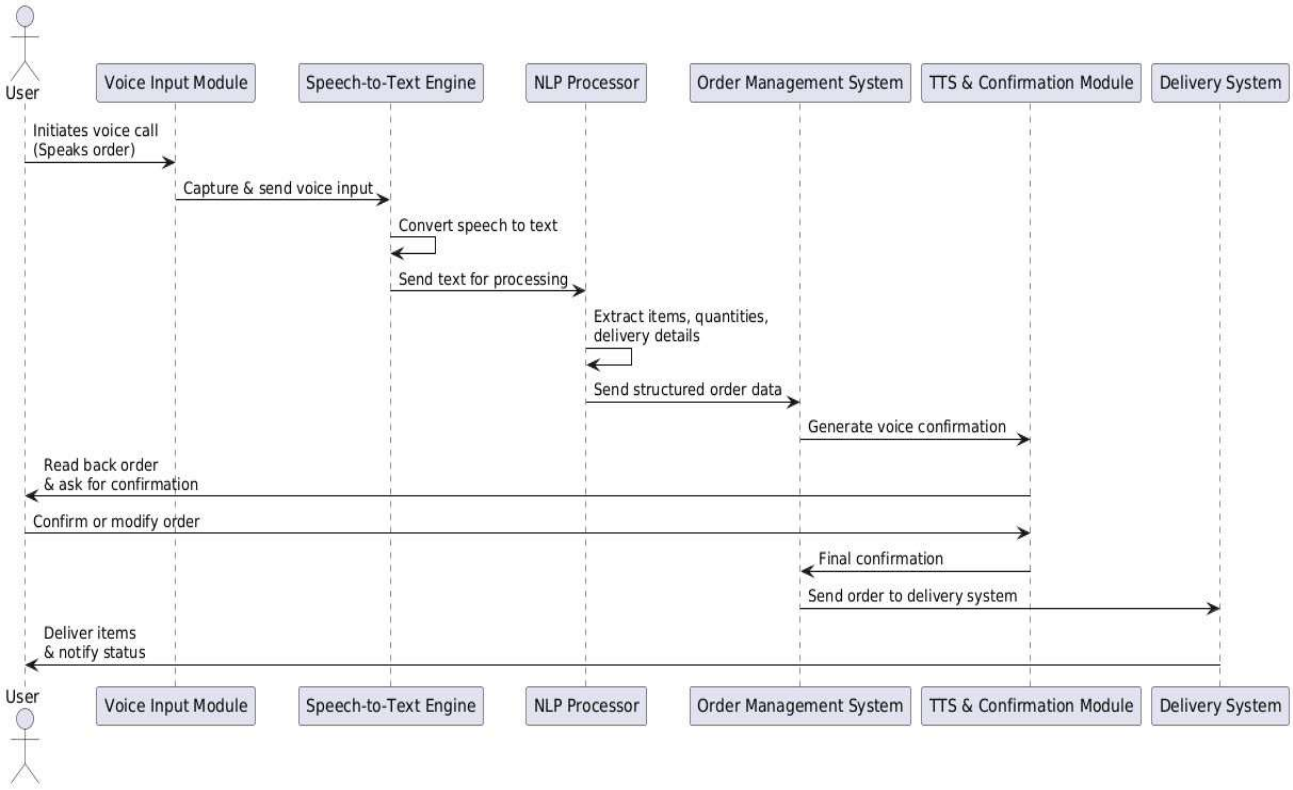


Fig. 3.3.4 Sequence Diagram

Key Interactions and Relationships

1. User and Voice Interface:
• The process begins when the user initiates a phone call to the voice-based ordering system.
• The system prompts the user to speak their ration order in their native language.
• The spoken order is captured by the **Voice Input Module** and sent to the **Speech-to-Text Module**.

2. Speech Recognition and Processing:
• The **Speech-to-Text Module** converts the user's spoken input into text.
• The text is forwarded to the **NLP Processing Module** for semantic analysis.

3. Order Extraction via NLP:
• The **NLP Module** analyzes the transcribed text to extract order-specific details such as:

- Item names (e.g., rice, sugar)
- Quantities (e.g., 2 kg, 1 liter)
- Delivery instructions (e.g., deliver today or tomorrow)
- Extracted data is structured and sent to the **Order Management Module**.

4. Order Construction and Formatting:
• The **Order Management Module** constructs a structured order object based on the parsed information.
• This includes item list, quantity, delivery mode, and time preference.
• The complete order is forwarded to the **Text-to-Speech Module** for voice confirmation.

5. Voice-Based Order Confirmation:
• The **Text-to-Speech (TTS) Module** reads the interpreted order back to the user in their selected language.
• The user is prompted to confirm or correct the order by saying "yes" or "no."
• The system waits for the user's response before proceeding.

6. Finalization and Delivery Notification:
• Once the user confirms the order, it is finalized by the **Order Management Module**.
• The order is sent to the **Delivery and Notification Module**, which notifies the appropriate vendor or delivery agent.
• An SMS or automated voice call is also sent to the user with the confirmed order details.

7. Notification to User:

• The user receives an SMS or voice call confirming the final order.
• This marks the end of the ordering cycle.
• The user may repeat the process for a new order if needed.

3.3.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in Fig. 3.3.5

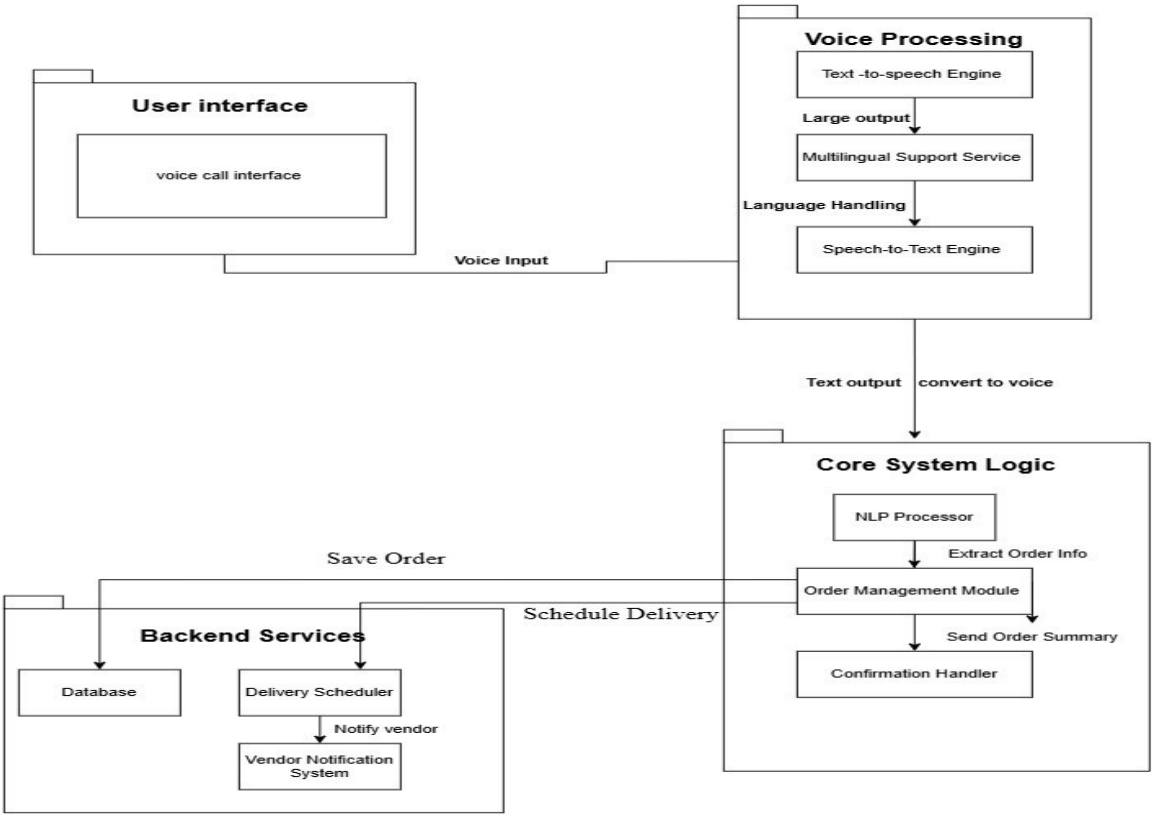


Fig. 3.3.5 Component Diagram

Main Components

1. User Interface:

• This component allows unlettered users to interact with the system using a basic phone via voice calls.

• Users can speak their ration order naturally in their preferred language.

• It serves as the entry point for voice-based order input and feedback delivery.

2. Voice Input Component:

• This component captures the user's voice input during the phone call.

• It ensures clear audio signal collection and handles any basic preprocessing.

• The captured voice is forwarded to the Speech-to-Text (STT) Component for transcription.

3. Speech-to-Text (STT) Component:

• This component converts the captured voice input into text using multilingual speech recognition.

• It supports various regional languages to ensure accessibility for diverse user groups.

• The converted text is passed on to the NLP Component for order interpretation.

4. NLP (Natural Language Processing) Component:

• This component processes the transcribed text to extract relevant details such as:

- Ration item names
- Quantities
- Delivery preferences
- The output is a structured order format that is forwarded to the Order Manager.

5. Order Management Component:

• This component validates and organizes the extracted order details.

• It constructs a formal order, handles logic for order correction or resubmission, and prepares it for voice confirmation.

• Once finalized, the order is stored and sent to the delivery backend.

6. Text-to-Speech (TTS) Component:

• This component reads the interpreted order back to the user in their preferred language for confirmation.

• It uses synthesized speech to communicate the item list, quantities, and delivery time to the user.

• It captures the confirmation (Yes/No) response and decides the next action accordingly.

7. Notification Component:

• This component handles post-confirmation communication.

• It generates a brief order summary and sends it to the user via SMS or voice call using the integrated messaging gateway.

• It also notifies the ration delivery vendor with the order details.

8. External Resources:

• **STT Engine (Google STT / Vosk):** Converts voice input into text in multiple languages.

• **TTS Engine (Google TTS / pyttsx3):** Converts structured text into spoken audio for feedback.

• **SMS Gateway:** Sends order confirmations and notifications to users.

• **Vendor Database:** Stores information about registered ration vendors and delivery handlers.

3.3.6 Deployment Diagram

The Deployment Diagram illustrates the physical distribution of software components across hardware modules in the Voice-Based Ration Ordering System. It outlines how each component interacts and functions in the real-world setup, particularly designed to support voice-based interactions for unlettered users.

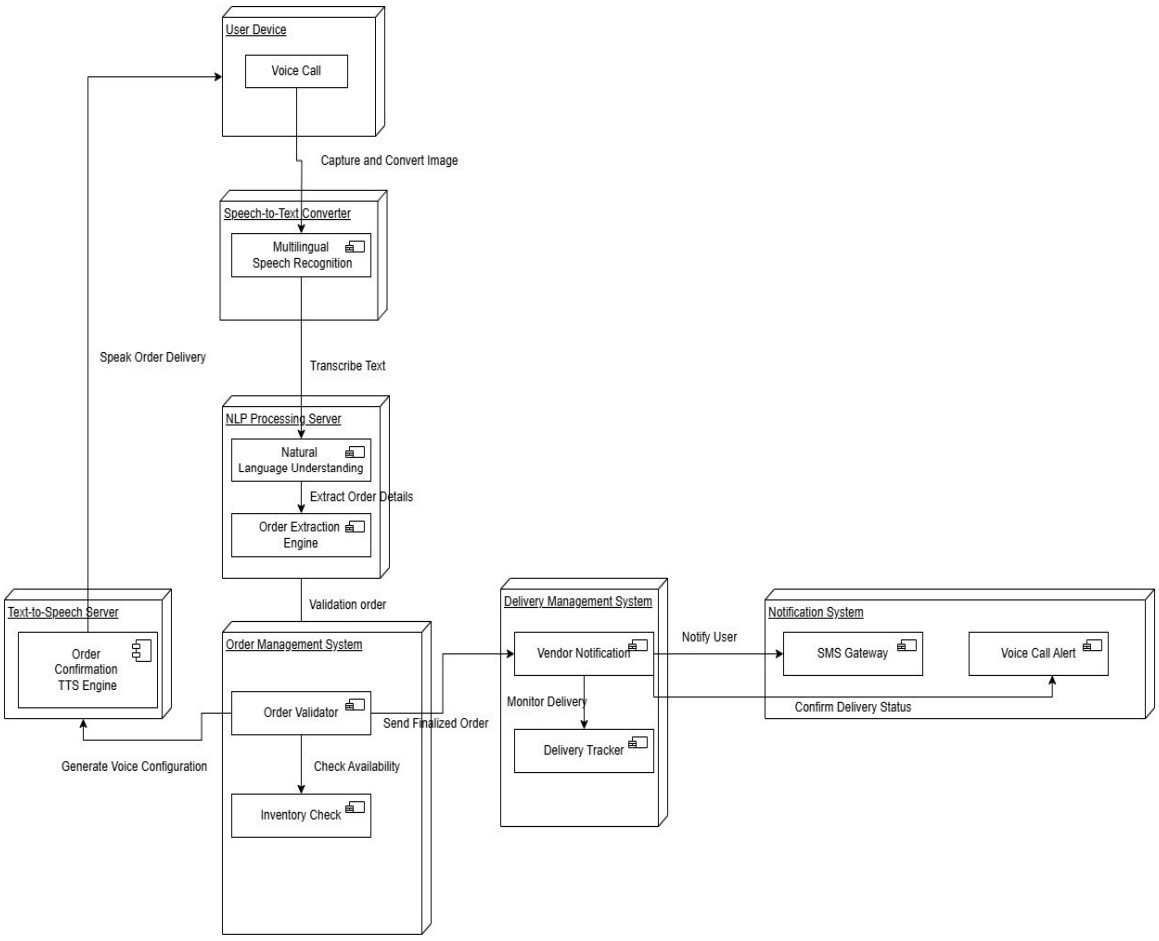


Fig. 3.3.6.1 Deployment Diagram

The system architecture is deployed across the following major hardware and software nodes:

- **User Device**:

  This node includes a basic mobile phone or any device capable of making a voice call. The user speaks their order through a voice call. The system captures and processes this voice input.

- **Speech-to-Text Converter**:

  This server component converts the captured voice signal into text using **Multilingual Speech Recognition**, enabling users to speak in their native language.

- **NLP Processing Server**:
  This server handles **Natural Language Understanding (NLU)** and extracts meaningful order details through the **Order Extraction Engine**. It interprets the transcribed text and passes structured order data forward.

- **Order Management System**:
  This node validates the extracted order using the **Order Validator** and checks item availability through an **Inventory Check** module. Once validated, it forwards the final order for fulfillment.

- **Text-to-Speech Server**:
  After processing, the system uses the **Order Confirmation TTS Engine** to generate a voice-based confirmation, allowing the user to hear and confirm their order.

- **Delivery Management System**:
  Responsible for forwarding the order to the vendor, this system monitors order status and updates delivery information using the **Delivery Tracker** and **Vendor Notification** modules.

- **Notification System**:
  This node sends delivery status updates and alerts to the user through SMS and **Voice Call Alerts**, ensuring that users without literacy skills can still receive and understand their order status.

This deployment structure ensures efficient voice-based interaction, seamless backend processing, and inclusive communication for unlettered users.

3.4 METHODOLOGY

3.4.1 Voice Input Collection and Processing

The system does not require a traditional dataset for training visual models, but instead relies on real-time **voice inputs** provided by users through basic mobile phones. The user speaks their ration order in their native language, which serves as the system's primary input.

To support diverse linguistic regions, the system integrates with **multilingual Speech-to-Text (STT) APIs** (e.g., Google STT, Vosk) that transcribe audio into text in multiple Indian languages like Telugu, Hindi, Tamil, etc. These inputs are dynamically processed without prior audio datasets but are handled using pre-trained ASR (Automatic Speech Recognition) models.

For improved understanding and robustness, the system includes error-handling mechanisms to correct common pronunciation variances, fillers, and background noise interference that may affect transcription.

Natural Language Processing (NLP) Pipeline

After converting speech to text, the system uses a **custom NLP pipeline** built using tools like spaCy or NLTK to extract structured information from user sentences. The pipeline performs:

- **Tokenization** – Splits the sentence into words.
- **Named Entity Recognition (NER)** – Identifies ration items, quantities, and delivery terms.
- **Dependency Parsing** – Understands the relationship between nouns (items) and their quantities or modifiers.
- **Intent Classification** – Ensures that the command is indeed an order and not just noise.

The NLP model is tuned using synthetically generated voice orders in multiple languages for training and testing.

Text-to-Speech (TTS) Confirmation Module

Once the structured order is formed, it is read back to the user for confirmation using a **Text-to-Speech (TTS)** engine (e.g., Google TTS or pyttsx3). The user confirms or corrects the order by saying "yes" or "no." This creates a closed feedback loop, increasing reliability and reducing misorders.

The TTS model is lightweight and optimized for real-time feedback, compatible with deployment on low-resource devices and servers.

Why These Models Are Used

- **STT (Speech-to-Text):** Enables conversion of natural speech into digital text across multiple regional languages with high accuracy.
- **NLP:** Converts unstructured user commands into structured JSON-like order data.
- **TTS (Text-to-Speech):** Ensures usability for unlettered individuals by enabling audible feedback and confirmation.

These modules work together to form a complete voice-based interaction system suitable for deployment in low-connectivity, rural environments without dependency on high-end hardware or screens.

Integration in the Project

In this project:

- The **STT component** captures and transcribes voice commands from unlettered users.
- The **NLP module** extracts key entities like item names, quantities, and delivery time.

- The **Order Management system** structures the extracted information into a valid order.
- The **TTS component** confirms the order with the user in their language.
- Upon confirmation, the order is forwarded to vendors and the user receives a confirmation via **SMS or voice call**.

This combination of real-time voice processing, multilingual capabilities, and structured order management makes the system **technically sound, linguistically inclusive, and practically viable** for rural deployment in ration delivery services.

4. CODE AND IMPLEMENTATION

4.1 CODE

App.py

```
from flask import Flask, render_template, request, redirect, url_for
import json
import os
from voice_pipeline import VoiceGrocerySystem


app = Flask(__name__)
system = VoiceGrocerySystem("Daily_Grocery_Prices.csv")


# Combined login and signup route
@app.route('/auth', methods=['GET', 'POST'])
def auth():
    if request.method == 'POST':
        action = request.form.get('action')


        if action == 'login':
            username = request.form['username']
            password = request.form['password']
            # 🔒 Dummy check (replace with real DB check)
            if username == 'admin' and password == '1234':
                return render_template('app.html')  # or return success JSON
            return "Invalid credentials", 401
```

```
        elif action == 'signup':
            # Extract signup details from form
            username = request.form['username']
            password = request.form['password']
            # TODO: Add logic to save user to DB or file
            return "Signup successful! Please log in."  # Or redirect to login

        else:
            return "Invalid action", 400

    # GET request renders combined login/signup page
    return render_template('auth.html')

@app.route('/')
def home():
    latest_bill = system.get_latest_bill()
    order_history = system.get_order_history()
    return render_template(
        'main.html',
        bill=latest_bill,
        history=order_history
    )

@app.route('/process_order', methods=['POST'])
```

```
def process_order():
    success = system.run_voice_order_pipeline()
    return redirect(url_for('home'))


if __name__ == '__main__':
    if not os.path.exists("order_history.json"):
        with open("order_history.json", "w") as f:
            json.dump([], f)
    app.run(debug=True)
```

order_history:


```
[
  {
    "timestamp": "2025-06-09T22:30:45.059496",
    "items": [
      {
        "item": "Made",
        "quantity": 10.0,
        "unit": "kg",
        "unit_price": 0.0,
        "total": 0.0,
        "confidence": 0.0,
        "status": "not_found"
      }
```

```
    ],
    "total": 0.0
  }
]
```

Voice_pipeline.py:

```
import speech_recognition as sr
from googletrans import Translator
from gtts import gTTS
import pygame
import os
import uuid
import spacy
import pandas as pd
from word2number import w2n
import inflect
import logging
import re
from fuzzywuzzy import fuzz
import json
from datetime import datetime
import time

# Configure logging
```

```
logging.basicConfig(level=logging.WARNING, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(__name__)


class VoiceGrocerySystem:

    def __init__(self, csv_path: str = "Daily_Grocery_Prices.csv"):

        self.csv_path = csv_path

        self.translator = Translator()

        self.p = inflect.engine()

        self.lang_map = {'english': 'en', 'hindi': 'hi', 'spanish': 'es', 'french': 'fr', 'german': 'de', 'tamil': 'ta', 'telugu': 'te', 'japanese': 'ja', 'korean': 'ko', 'chinese': 'zh-cn', 'portuguese': 'pt', 'russian': 'ru', 'arabic': 'ar', 'italian': 'it'}

        self.stop_words = {'en': ['done', 'complete', 'finished', 'stop', 'enough'], 'hi': ['हो गया', 'पूरा', 'बस'], 'te': ['చాలు', 'అంతే'], 'ta': ['முடிந்தது', 'போதும்']}

        self.unit_words = {'weight': ['kg', 'kilogram', 'g', 'gram', 'pound', 'lb'], 'volume': ['litre', 'liter', 'l', 'ml'], 'count': ['piece', 'pack', 'dozen', 'bottle', 'box']}

        self._init_components()

        self.last_order_summary = []

        self.last_order_total = 0

        self.order_history = []

        self._load_order_history()


    def _init_components(self):

        try:

            self.nlp = spacy.load("en_core_web_sm")

        except OSError:
```

```
            logger.error("SpaCy model not found. Install: python -m spacy download
en_core_web_sm")
            raise
        try:
            if os.path.exists(self.csv_path):
                self.price_data = pd.read_csv(self.csv_path)
                self.price_data['item_normalized'] = self.price_data['item'].str.lower().str.strip()
            else:
                logger.warning(f"Price data file not found: {self.csv_path}")
                self.price_data = pd.DataFrame(columns=['item', 'unit', 'price_per_unit'])
        except Exception as e:
            logger.error(f"Error loading price data: {e}")
            self.price_data = pd.DataFrame(columns=['item', 'unit', 'price_per_unit'])
        try:
            pygame.mixer.init()
        except Exception as e:
            logger.warning(f"Audio init failed: {e}")


    def _load_order_history(self):
        if os.path.exists("order_history.json"):
            try:
                with open("order_history.json", "r") as f:
                    self.order_history = json.load(f)
            except Exception as e:
                logger.error(f"Error loading order history: {e}")
```

```
    def speak(self, text: str, lang: str = 'en', cleanup: bool = True) -> bool:
        if not text.strip(): return False
        try:
            filename = f"temp_audio_{uuid.uuid4().hex[:8]}.mp3"
            gTTS(text=text, lang=lang, slow=False).save(filename)
            pygame.mixer.music.load(filename)
            pygame.mixer.music.play()
            while pygame.mixer.music.get_busy(): time.sleep(0.1)
            if cleanup and os.path.exists(filename): os.remove(filename)
            return True
        except Exception as e:
            logger.error(f"TTS error: {e}")
            return False


    def listen(self, language: str = 'en', timeout: int = 10, phrase_time_limit: int = 5) -> str:
        r = sr.Recognizer()
        r.energy_threshold = 300
        r.dynamic_energy_threshold = True
        with sr.Microphone() as source:
            try:
                r.adjust_for_ambient_noise(source, duration=1)
                audio = r.listen(source, timeout=timeout,
phrase_time_limit=phrase_time_limit)
                return r.recognize_google(audio, language=language).lower().strip()
```

```
            except (sr.UnknownValueError, sr.WaitTimeoutError): return ""
            except sr.RequestError as e: logger.error(f"SR error: {e}"); return ""

    def detect_language(self, text: str) -> str:
        if not text.strip(): return 'en'
        try:
            detected_lang = self.translator.translate(text, dest='en').src
            for lang_name, lang_code in self.lang_map.items():
                if fuzz.ratio(detected_lang, lang_name) > 70: return lang_code
            return 'en'
        except Exception as e: logger.error(f"Lang detection error: {e}"); return 'en'

    def extract_items_enhanced(self, text: str) -> list[dict]:
        items = []
        patterns = [r'(\d+(?:\.\d+)?)\s*(?:kg|kilogram(?:s)?|kilo)\s+(?:of\s+)?(\w+)',
r'(\d+(?:\.\d+)?)\s*(?:g|gram(?:s)?)\s+(?:of\s+)?(\w+)',
r'(\d+(?:\.\d+)?)\s*(?:litre|liter(?:s)?|l)\s+(?:of\s+)?(\w+)',
r'(\d+(?:\.\d+)?)\s*(?:packet|packets|pack|packs)\s+(?:of\s+)?(\w+)',
r'(\d+(?:\.\d+)?)\s*(?:dozen)\s+(\w+)', r'(\d+(?:\.\d+)?)\s+(\w+)']
        for pattern in patterns:
            for match in re.findall(pattern, text, re.IGNORECASE):
                try:
                    qty, item_candidate = float(match[0]), match[1].lower()
                    if self._is_unit_word(item_candidate): continue
                    items.append({"item": self.singularize(item_candidate), "quantity": qty,
"unit": self._extract_unit_from_pattern(pattern), "confidence": 0.9})
                except (ValueError, IndexError): pass
```

```
        return self._deduplicate_items(items or self._extract_with_spacy(self.nlp(text)))


    def _extract_with_spacy(self, doc) -> list[dict]:
        items, current_quantity, current_unit = [], None, None
        for token in doc:
            if token.pos_ == "NUM" or token.like_num:
                try: current_quantity = w2n.word_to_num(token.text)
                except: current_quantity = float(token.text) if token.text.replace('.', '',
1).isdigit() else None
            elif self._is_unit_word(token.text.lower()): current_unit = token.text.lower()
            elif token.pos_ in ["NOUN", "PROPN"] and current_quantity is not None:
                items.append({"item": self.singularize(token.text.lower()), "quantity":
current_quantity, "unit": current_unit or "", "confidence": 0.7})
                current_quantity, current_unit = None, None
        return items
    def _is_unit_word(self, word: str) -> bool:
        return any(word in units for units in self.unit_words.values())
    def _extract_unit_from_pattern(self, pattern: str) -> str:
        if 'kg' in pattern: return 'kg'
        if 'g' in pattern: return 'g'
        if 'litre' in pattern or 'liter' in pattern: return 'litre'
        if 'packet' in pattern or 'pack' in pattern: return 'packet'
        if 'dozen' in pattern: return 'dozen'
        return ''
    def _deduplicate_items(self, items: list[dict]) -> list[dict]:
```

```
        item_dict = {}; [item_dict.setdefault(f"{item['item']}_{item['unit']}",
item)['quantity'] += item['quantity'] for item in items if f"{item['item']}_{item['unit']}" in
item_dict or not item_dict.setdefault(f"{item['item']}_{item['unit']}", item.copy())];
return list(item_dict.values())


    def find_price_fuzzy(self, item: str, unit: str) -> tuple[float, float]:

        if self.price_data.empty: return 0.0, 0.0

        item_norm = self.singularize(item.lower())

        exact_match = self.price_data[(self.price_data['item_normalized'] == item_norm) &
(self.price_data['unit'].str.lower().str.contains(unit.lower() if unit else '', na=False))]

        if not exact_match.empty: return exact_match.iloc[0]['price_per_unit'], 1.0

        best_match, best_score = None, 0

        for _, row in self.price_data.iterrows():

            score = fuzz.ratio(item_norm, row['item_normalized'])

            if score > best_score and score > 70: best_score, best_match = score, row

        return (best_match['price_per_unit'], best_score / 100) if best_match is not None
else (0.0, 0.0)

    def calculate_total_enhanced(self, extracted_items: list[dict]) -> tuple[list[dict], float]:

        summary, grand_total = [], 0

        for entry in extracted_items:

            item, quantity, unit = entry["item"], entry["quantity"], entry["unit"]

            price_per_unit, confidence = self.find_price_fuzzy(item, unit)

            total = quantity * price_per_unit

            grand_total += total

            summary.append({"item": item.title(), "quantity": quantity, "unit": unit,
"unit_price": price_per_unit, "total": total, "confidence": confidence, "status": "found" if
price_per_unit > 0 else "not_found"})

        return summary, grand_total
```

```
    def display_summary_table(self, summary: list[dict], grand_total: float):
        print("\n" + "="*50 + "\n🛒 ORDER SUMMARY\n" + "="*50)
        print(f"{'Item':<15} {'Qty':<6} {'Unit':<8} {'Price':<8} {'Total':<8} {'Status':<6}")
        print("-"*50)
        for item in summary:
            print(f"{item['item']:<15} {item['quantity']:<6} {item['unit']:<8} ₹{item['unit_price']:<7.2f} ₹{item['total']:<7.2f} {'✅' if item['status'] == 'found' else '❌'}")
        print("-"*50 + f"\n{'GRAND TOTAL:':<37} ₹{grand_total:.2f}\n" + "="*50)
    def save_order_history(self, summary: list[dict], total: float):
        self.order_history.append({"timestamp": datetime.now().isoformat(), "items": summary, "total": total})
        try:
            tmp_file = "order_history.tmp.json"
            with open(tmp_file, "w") as f: json.dump(self.order_history, f, indent=2)
            os.replace(tmp_file, "order_history.json")
        except Exception as e: logger.error(f"Error saving history: {e}")
    def singularize(self, word: str) -> str:
        return self.p.singular_noun(word) or word
    def run_voice_order_pipeline(self) -> bool:
        self.speak("Hello! Please say your preferred language.", lang='en')
        lang_code = self.detect_language(self.listen(timeout=15) or 'english')
        self.speak(self.translator.translate("Hello! Welcome to our voice grocery ordering system. Please tell me your complete order. Say 'done' when finished.", dest=lang_code).text, lang=lang_code)
        full_order = ""
        for _ in range(3):
```

```
            order_text = self.listen_until_done(lang_code)
            if order_text.strip(): full_order = order_text; break
            self.speak(self.translator.translate("I didn't catch that. Please try again.", dest=lang_code).text, lang=lang_code)
        if not full_order.strip(): self.speak(self.translator.translate("No order received. Please try again later.", dest=lang_code).text, lang=lang_code); return False
        translated_order = self.translator.translate(full_order, dest='en').text if lang_code != 'en' else full_order
        extracted = self.extract_items_enhanced(translated_order)
        if not extracted: self.speak(self.translator.translate("Sorry, I couldn't understand your order.", dest=lang_code).text, lang=lang_code); return False
        summary, grand_total = self.calculate_total_enhanced(extracted)
        self.last_order_summary, self.last_order_total = summary, grand_total
        self.display_summary_table(summary, grand_total)
        self.save_order_history(summary, grand_total)
        summary_text = f"Your order includes: {', '.join([f'{item['quantity']} {item['unit']} {item['item']}' for item in summary if item['status'] == 'found'])}. Total cost is {grand_total:.2f} rupees."
        if any(item['status'] == 'not_found' for item in summary): summary_text += f" Note: Prices not found for {', '.join([item['item'] for item in summary if item['status'] == 'not_found'])}."
        self.speak(self.translator.translate(summary_text, dest=lang_code).text, lang=lang_code)
        return True
    def listen_until_done(self, lang_code: str) -> str:
        collected, stop_words = [], self.stop_words.get(lang_code, self.stop_words['en'])
        while True:
            phrase = self.listen(language=lang_code, timeout=10)
```

```
            if not phrase: continue
            if any(stop.lower() in phrase.lower() for stop in stop_words): break
            collected.append(phrase)
            if len(collected) % 3 == 0: self.speak(self.translator.translate("Continue...", dest=lang_code).text, lang=lang_code)
        return " ".join(collected)
    def get_last_order(self) -> tuple[list[dict], float]: return self.last_order_summary, self.last_order_total
    def get_order_history(self) -> list[dict]: return self.order_history
    def get_latest_bill(self): return {"timestamp": datetime.now().isoformat(), "items": self.last_order_summary, "total": self.last_order_total} if self.last_order_summary else {"timestamp": None, "items": [], "total": 0.0}
```

4.2 IMPLEMENTATION

Create a directory folder as follows and place the relevant modules, scripts, and templates accordingly:

4.2.1 Directory Folder

Sql

VOICE-RATION-SYSTEM/

```
│
├── app.py
│
├── static/
│   ├── audio/
│   └── instructions.mp3
│
├── templates/
│   └── order_interface.html
│
├── modules/
│   ├── speech_to_text.py
│   ├── text_to_speech.py
│   ├── nlp_extraction.py
│   ├── order_handler.py
│   └── sms_notification.py
│
├── orders/
│   └── confirmed_orders.json
```

4.2.2 Speech & NLP Pipeline Development (Google Colab)

The **voice processing and NLP logic** were initially developed and tested on Google Colab using pre-recorded sample commands in multiple languages. The following steps outline the pipeline:

1. **Voice Input Collection & Preprocessing**

   - Audio files of user voice commands were captured or simulated.
   - Audio was converted to text using multilingual **Google Speech-to-Text API**.
   - Test data included commands in Telugu, Hindi, and English.

2. **NLP Model for Order Extraction**

   - Using **spaCy** and **NLTK**, models were developed to extract ration item names, quantities, and delivery dates.
   - A JSON-like structure was created:

```
{
 "items": [{"name": "rice", "quantity": "2 kg"}, {"name": "sugar", "quantity": "1 kg"}],
                "delivery": "tomorrow"}
```

3. **TTS Confirmation Flow**

   - The extracted order was converted back to speech using **Google TTS**.
   - Example spoken output: "You ordered 2 kg rice and 1 kg sugar for tomorrow. Say 'yes' to confirm."

4. **Testing & Export**

   - Modules were tested with multiple accents and dialects.
   - Final Python scripts were exported from Colab and placed into the local Flask application.

4.2.3 Integration and Deployment

Once each module was developed and tested, they were integrated into a **Flask web application** via app.py. The backend manages the interaction flow between the user and the system over voice, with real-time confirmation and SMS-based feedback.

Additional integrations:

- **Text-to-Speech (TTS):** Converts structured order data into regional-language audio for playback via phone or web.
- **Speech-to-Text (STT):** Supports multiple languages using Google's API or Vosk for offline capability.
- **SMS Notification:** Twilio API is used to send a confirmation message containing the order details to the user's phone.

4.2.4 User Interaction Flow

1. The user initiates a voice call to the system and speaks the order.
2. The voice is captured and transcribed into text using the STT module.
3. NLP extracts item names, quantities, and delivery preferences.
4. The order is read back to the user for confirmation via Text-to-Speech.
5. If the user confirms, the order is stored and sent to the delivery vendor.
6. An SMS is sent to the user confirming the order, completing the interaction.

5. TESTING

5.1 INTRODUCTION TO TESTING

Testing is a critical phase in software development that ensures the correct functionality, usability, performance, and reliability of an application under various conditions. It verifies that the final system meets the intended specifications and behaves consistently, both with expected and unexpected user inputs. Effective testing minimizes the risk of system failures and enhances the user experience by identifying bugs or inconsistencies early in the development lifecycle.

In this project, the Voice-Based Grocery Delivery System, testing played a vital role in ensuring that voice recognition, multilingual support, order processing, and feedback mechanisms all operated as intended. The objective was to validate that each component—from voice input and natural language processing to product availability checks and order confirmation—functioned seamlessly and contributed to a smooth and user-friendly grocery ordering experience.

The test cases were designed to verify the core functionalities and workflow of the application, including:

- Accurate recognition and interpretation of voice commands in multiple languages
- Correct transition between application pages (home, order, confirmation, order history, etc.)
- Reliable processing and confirmation of single and multiple item orders
- Handling of unavailable or ambiguous product requests
- Proper feedback to the user in case of errors or invalid commands
- Secure storage and retrieval of order history and user preferences
- Robustness against background noise and unexpected inputs
- System response to backend server downtime or technical issues

5.2 TEST CASES:

Table 5.1 Test Cases of Rhythm Restore

Test Case ID	Test case Name	Test Description	Expected Output	Actual Output	Remarks
TC_01	Valid Single Item Order	User orders a single available item via voice.	System recognizes item and confirms order with price.	System recognizes item and confirms order with price.	Successful
TC_02	Valid Multiple Items Order	User orders multiple available items in one command.	System recognizes all items and confirms order with total price.	System recognizes all items and confirms order with total price.	Successful
TC_03	Unavailable Item Order	User orders an item not in the inventory.	System informs user that item is unavailable.	System informs user that item is unavailable	Successful
TC_04	Ambiguous Item Name	User says an item name that could refer to multiple products	System asks user to clarify the item.	User clarifies the item	Successful
TC_05	Background Noise Interference	User gives command with significant background noise.	System requests user to repeat the command or accurately filters the noise	System requests user to repeat the command or accurately filters the noise	Successful
TC_06	Non-English Language Input	User orders items using a supported	System recognizes language,	System recognized the	Successful

		non-English language.	translates, and confirms order.	language and translated it	
TC_07	Order Cancellation	User cancels an order after placing it.	System confirms order cancellation and updates order status.	System confirmed the order and updated the status.	Successful
TC_08	Price Inquiry	User asks for the price of a specific item.	System provides the current price of the requested item.	System provided the current price of the item.	Successful
TC_09	Invalid Command	User gives a command unrelated to grocery ordering.	System informs user the command is invalid and provides usage guidance.	System informed the user that it is invalid.	Successful
TC_10	System Downtime Handling	User tries to place an order when the backend server is down.	System informs user of technical issues and suggests trying again later.	System informs user of technical issues and suggests trying again later.	Successful

6. RESULTS

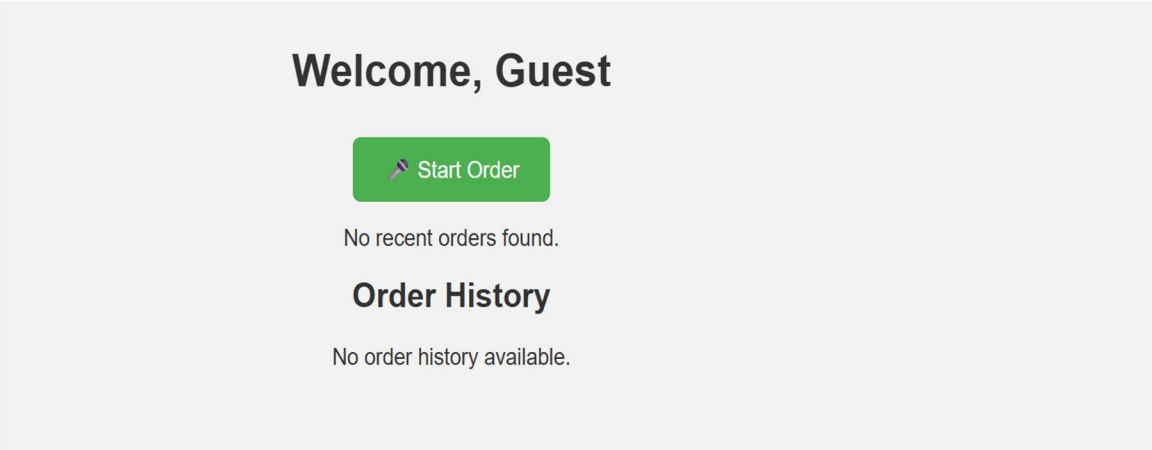


Fig. 6.1 Initial Page

The image shows the initial user interface of the Voice Grocery Delivery System for a guest user. Upon accessing the system, the user is greeted with a welcome message and presented with a prominent green "Start Order" button, which initiates the voice-based ordering process. Since the user has not placed any orders yet, the interface displays a message indicating that no recent orders are found. Additionally, the "Order History" section confirms that there is no order history available for the guest user.

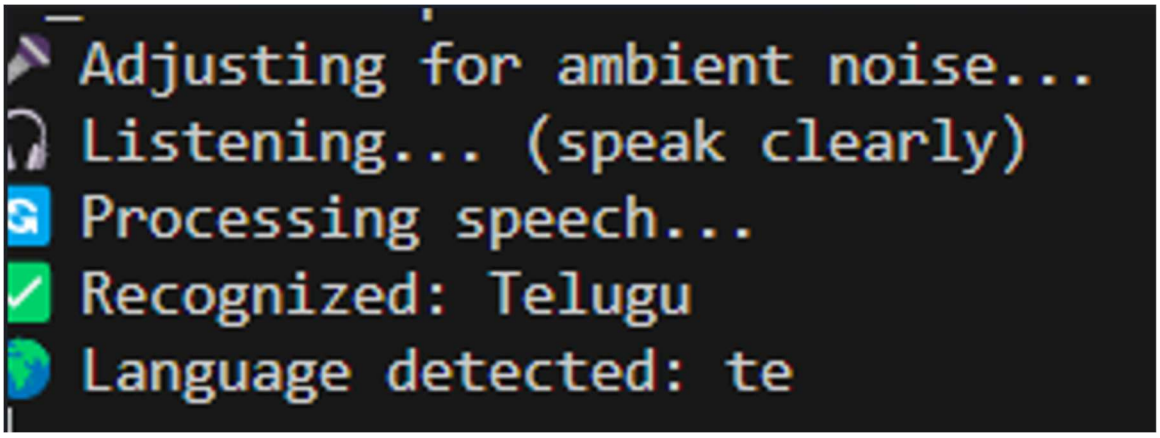


Fig . 6.2 Language recognition

The image displays a console output from the Voice Grocery Delivery System during the speech recognition process. The system first adjusts for ambient noise to ensure

accurate voice input. It then enters listening mode, prompting the user to speak clearly. After capturing the audio, the system processes the speech and successfully recognizes the spoken language as Telugu. The detected language code "te" is also displayed, confirming that the system can accurately identify and process regional languages for voice-based grocery ordering. This step is crucial for enabling multilingual support and improving accessibility for users who prefer to interact in their native language.

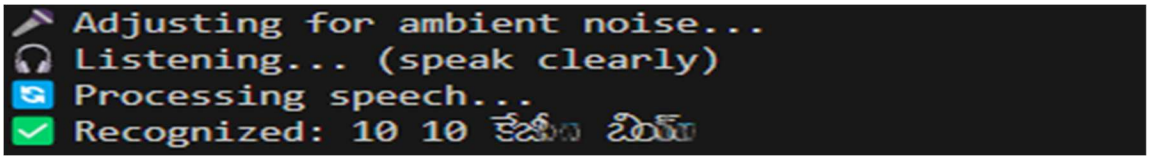


Fig .6.3 User Input taking(1)

The image displays the terminal output of the voice-based grocery delivery system, where it adjusts for ambient noise and listens for user input. It then processes the captured speech and successfully recognizes the spoken phrase.

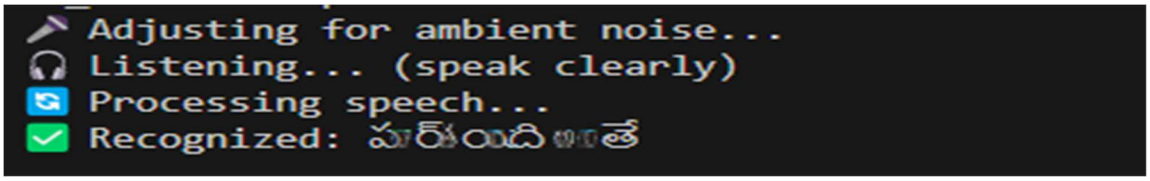


Fig . 6.4 User Input taking(2)

The image shows a terminal output from the voice-based grocery delivery system, capturing the speech recognition process. It successfully adjusts for ambient noise, listens to the user's voice input, and processes the speech.

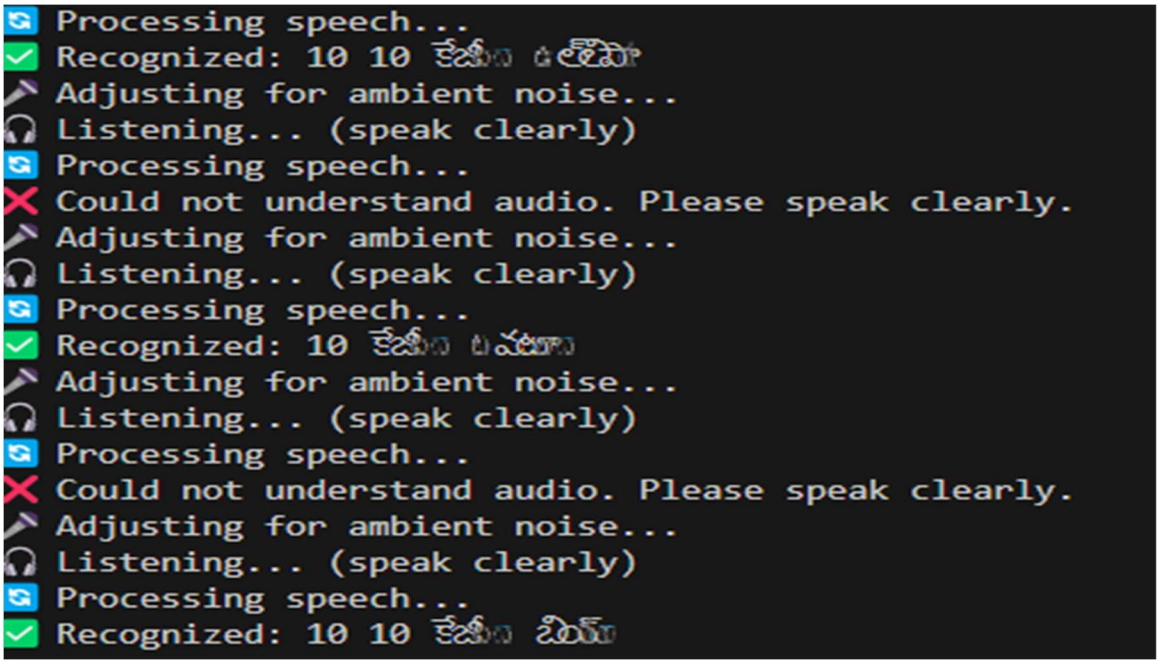


Fig . 6.5 User Input taking(3)

The image captures a series of terminal outputs from the voice-based grocery delivery system during multiple speech recognition attempts. The system successfully recognizes Telugu inputs like 10 kg onion after adjusting for ambient noise and processing speech.

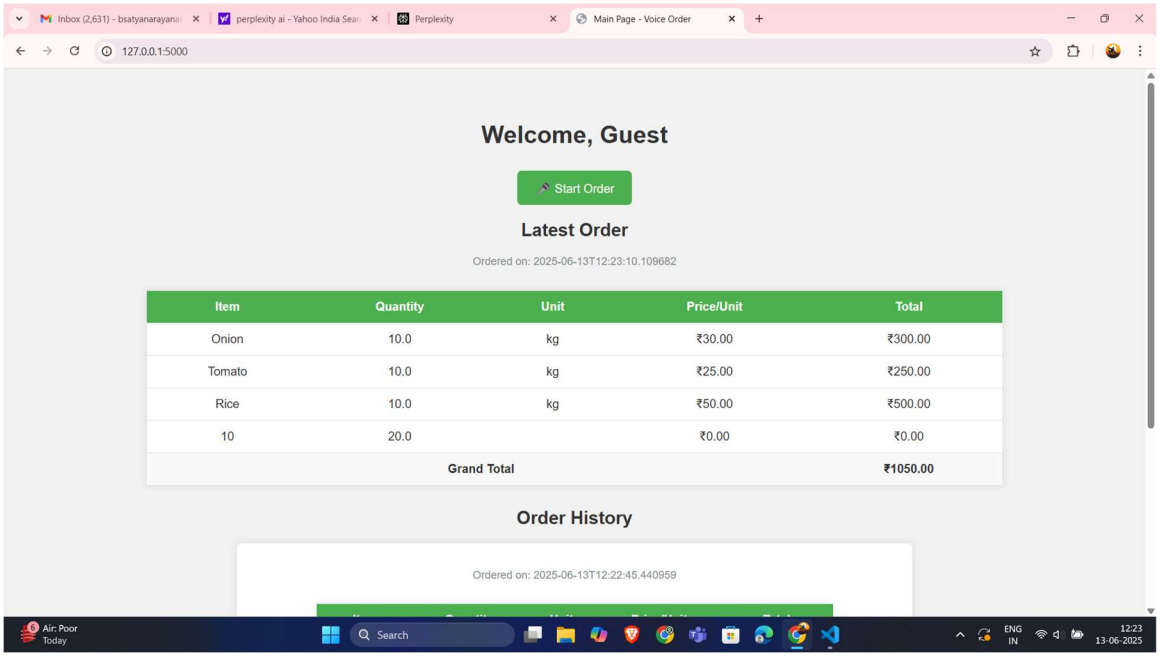


Fig . 6.6 Main Order Page

The image shows the user interface of the voice-based grocery delivery system running on a local server (127.0.0.1:5000). It welcomes the user and displays a summary of the latest order, which includes items like Onion, Tomato, and Rice with their quantities, units, and total prices. The system calculates a grand total of ₹1050.00 and maintains an order history for reference.

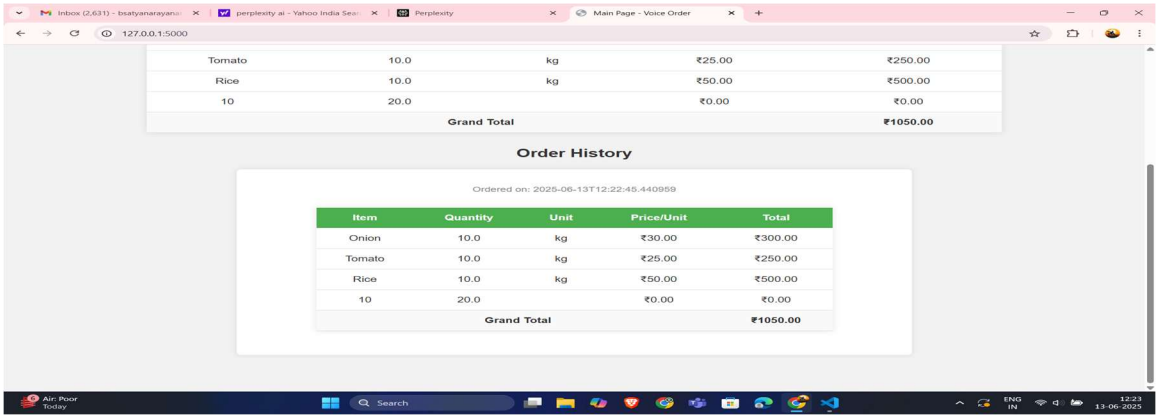


Fig . 6.7 Order history

The image displays the "Order History" section of the voice-based grocery delivery system's web interface. It shows a previously placed order including items like Onion, Tomato, and Rice with their respective quantities, units, price per unit, and total cost.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The Voice-Based Ration Ordering System for Unlettered Individuals successfully showcases how speech technology and AI-driven Natural Language Processing can bridge the gap between essential public services and marginalized populations. By enabling users to place ration orders through simple voice commands in their native languages, the system removes major barriers posed by illiteracy, digital inaccessibility, and regional language diversity.

Through its integration of **Speech-to-Text (STT)**, **Text-to-Speech (TTS)**, **NLP-based entity extraction**, and **multilingual SMS confirmation**, the system offers a **user-centric, inclusive, and automated solution** for last-mile delivery of ration goods. Designed for accessibility, especially in rural and semi-urban areas with limited internet and smartphone penetration, the solution promotes **autonomy, dignity, and trust** in public service delivery.

The architecture's modularity, multilingual support, and voice-based feedback loop reflect a strong focus on **practical deployment**, **low-tech compatibility**, and **scalability**—making it a robust framework for expanding public welfare services in underrepresented communities.

7.2 FUTURE ENHANCEMENT

Several forward-looking enhancements can extend the utility and inclusiveness of the Voice-Based Ration Ordering System:

- **Offline Voice Command Support**: Incorporating embedded, lightweight speech models (e.g., Vosk or DeepSpeech) for offline use in remote villages without network access.
- **Smart IVR System**: Implementing an Interactive Voice Response system that can dynamically guide users through menu-based voice ordering using number key inputs or yes/no replies.

- **Aadhaar-based Verification**: Integration with Aadhaar authentication can secure the user identity and improve service personalization.
- **Support for Grievance Redressal**: Adding modules for users to raise complaints or request help via voice could improve transparency and service quality.
- **Regional Product Recommendations**: Using simple AI models to suggest essential items based on previous orders, season, or area trends can personalize the ordering experience.
- **Mobile App Companion**: For semi-literate users or family members, a basic companion app with icons, voice prompts, and regional language support can improve system reach.
- **Voice Biometrics**: Future versions can incorporate voice recognition to authenticate users securely without PINs or passwords.
- These enhancements will not only strengthen the system's technical capabilities but also ensure that it evolves as a **scalable, inclusive, and socially impactful public service platform**.

REFERENCES

[1] Medhi, I., Patnaik, S., Brunskill, E., Gautama, S. N. N., Thies, W., & Toyama, K.(2011). Designing mobile interfaces for novice and low-literacy users. ACM Transactions on Computer-Human Interaction (TOCHI), 18(1), 1–28.

[2] Bano, S., Jithendra, P., Niharika, G. L., & Yalavarthi, S. (2020, November). Speech to text translation enabling multilingualism. In 2020 IEEE International Conference for Innovation in Technology (INOCON) (pp. 1–5). IEEE.

[3] Islam, M. N., Khan, N. I., Inan, T. T., & Sarker, I. H. (2023). Designing user interfaces for illiterate and semi-literate users: A systematic review and future research agenda. SAGE Open, 13(2), 1–14.

[4] Restyandito, R., & Chan, A. H. S. (2014, December). A review on voice user interface for illiterate people. In SEANES 2014: 3rd South East Asian Network of Ergonomics Societies Conference.

[5] Saloni, & Singh, W. (2020). Multilingual speech to text conversion – A review.EasyChair Preprint No.3064.

[6] Edim Azom Emmanuel & Muyingi Hippolyte N. (2014). Speech user interface for low literacy users of ICT services. European Journal of Computer Science and Information Technology, 2(1), 18-29.

[7] Sadeeq Jan, Imran Maqsood, Iftikhar Ahmad, Majid Ashraf, Fazal Qudus Khan, & Muhammad Imran Khan Khalil. (2019). A systematic feasibility analysis of user interfaces for illiterate users. Pakistan Academy of Sciences A: Physical and Computational Sciences, 56(4), 75–91.

[8] A Systematic Survey of Multilingual Speech Transcription and Translation Systems. (2023). International Journal of Advanced Research in Science, Communication and Technology, 10(1), 1–8.

[9] Azom Emmanuel, E., & Muyingi, H. N. (2013). A mobile user interface for low-literacy users in rural communities. Global Journal of Mathematical Sciences, 12(1), 1–11.

[10] Medhi, I., Jain, M., Tewari, A., & Toyama, K. (2011). User interface design for low-literate and novice users. Foundations and Trends® in Human–Computer Interaction, 4(1), 1–79.