Product Design

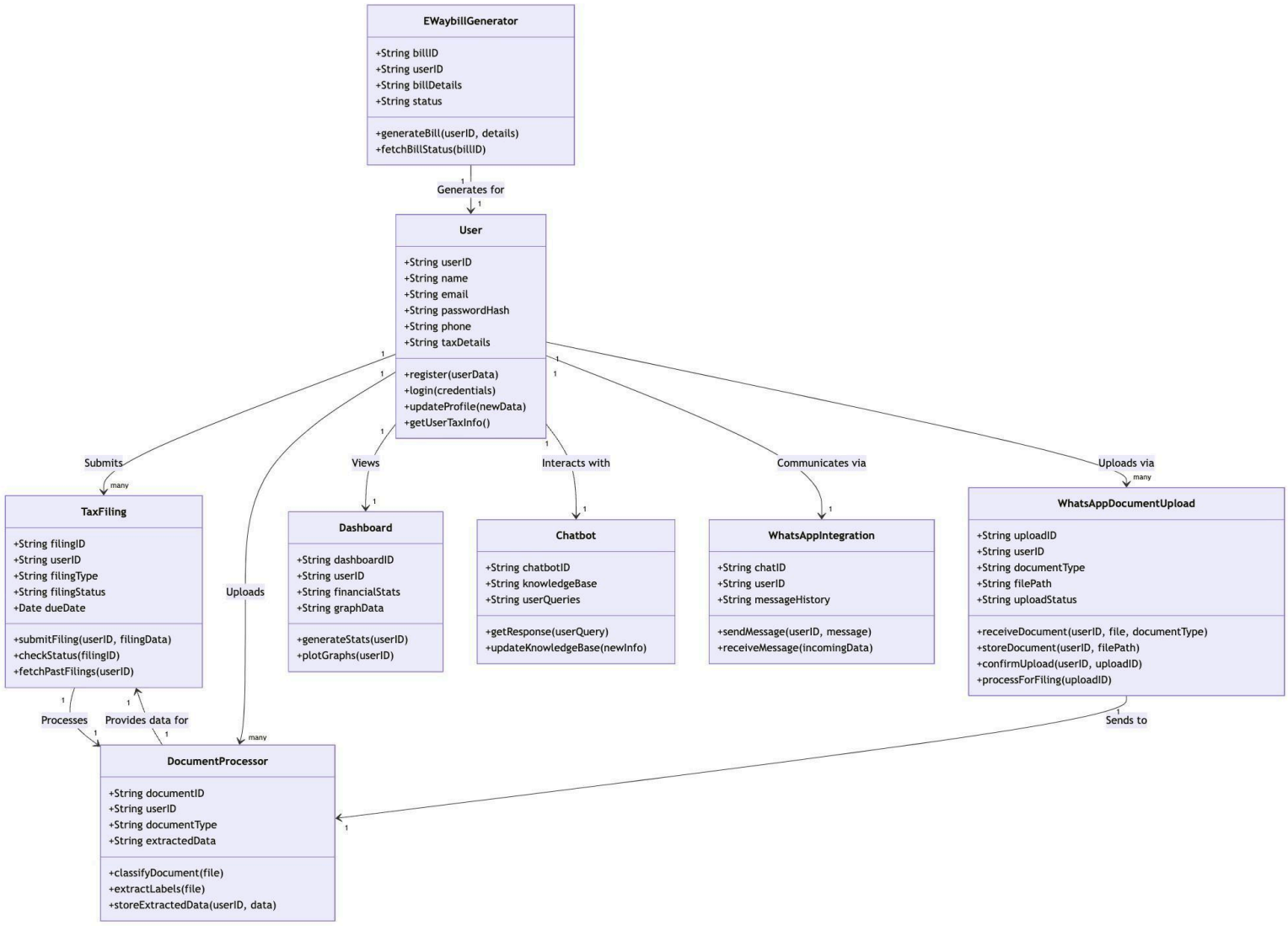
Team Number - 2

Team Name - Finease

TEAM MEMBERS

- **KUSHAGRA TRIVEDI**
- **SWAM SINGLA**
- **RONAK GAUR**
- **NIDHI VAIDYA**
- **AYUSH KUMAR GUPTA**

Design Model



Class No. & Name	Class State (Information Maintained) & Behavior (Methods Implemented)

1. User	**State:** - userID (Unique identifier) - name - email - passwordHash - phone - taxDetails **Behavior:** - register(userData) - login(credentials) - updateProfile(newData) - getUserTaxInfo()

2. TaxFiling	**State:** - filingID - userID - filingType (GST, ITR, PF) - filingStatus - dueDate **Behavior:** - submitFiling(userID, filingData) - checkStatus(filingID) - fetchPastFilings(userID)
3. DocumentProcessor	**State:** - documentID - userID - documentType (GST, PF, ITR) - extractedData **Behavior:** - classifyDocument(file) - extractLabels(file) - storeExtractedData(userID, data)

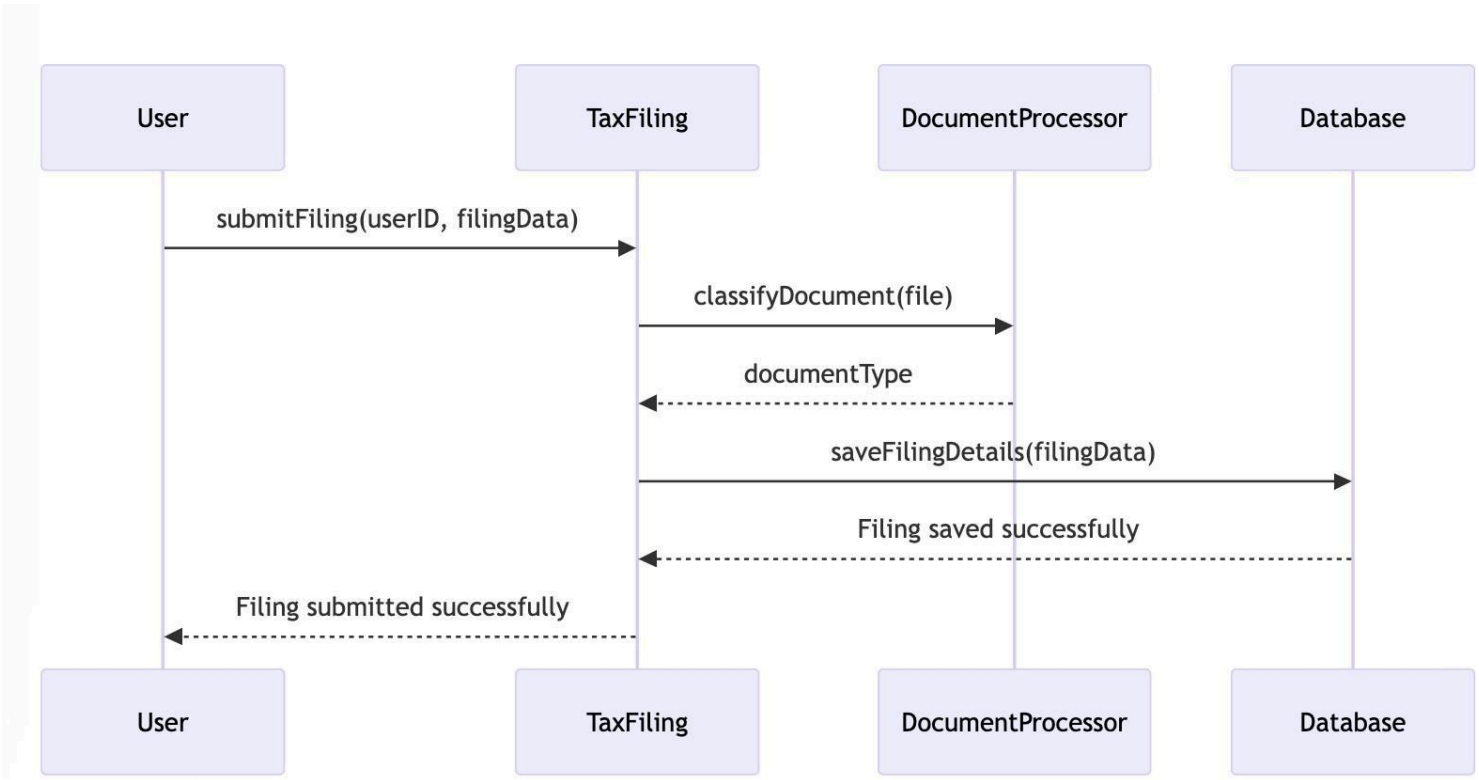
4. Chatbot	**State:** - chatbotID - knowledgeBase - userQueries **Behavior:** - getResponse(userQuery) - updateKnowledgeBase(newInfo)
5. EWaybillGenerator	**State:** - billID - userID - billDetails - status **Behavior:** - generateBill(userID, details) - fetchBillStatus(billID)
6. WhatsAppIntegration	**State:** - chatID - userID - messageHistory **Behavior:** - sendMessage(userID, message) - receiveMessage(incomingData)

7. WhatsAppDocumentUpload	**State:** - uploadID - userID - documentType (GST, ITR, PF) - filePath - uploadStatus **Behavior:** - receiveDocument(userID, file, documentType) - storeDocument(userID, filePath) - confirmUpload(userID, uploadID) - processForFiling(uploadID)

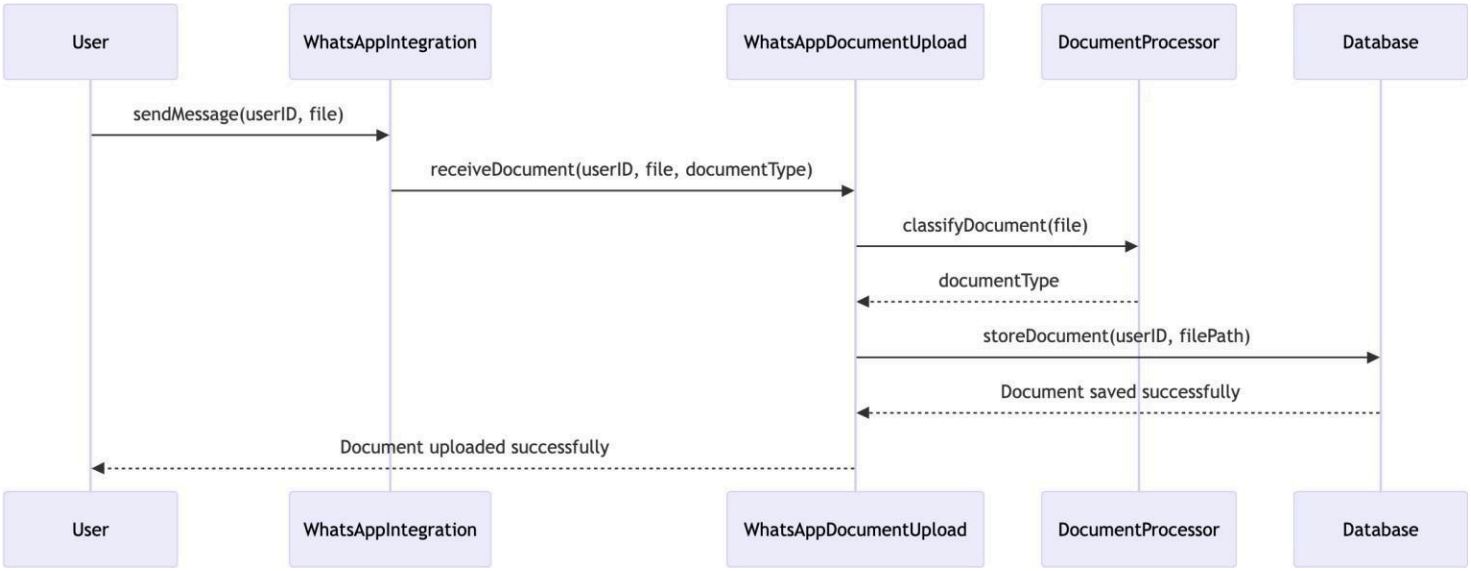
8. Dashboard	**State:** - dashboardID - userID - financialStats - graphData **Behavior:** - generateStats(userID) - plotGraphs(userID)

Sequence Diagram(s)

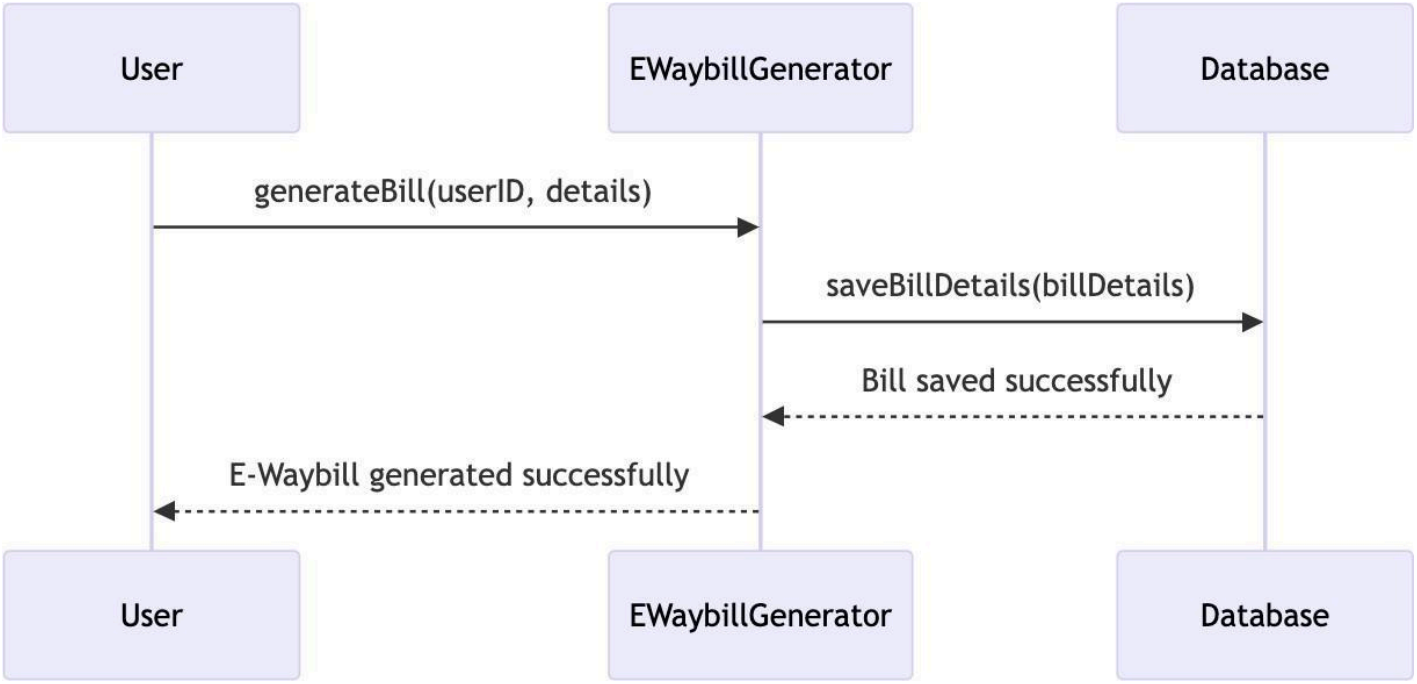
TAX FILING SEQUENCE DIAGRAM



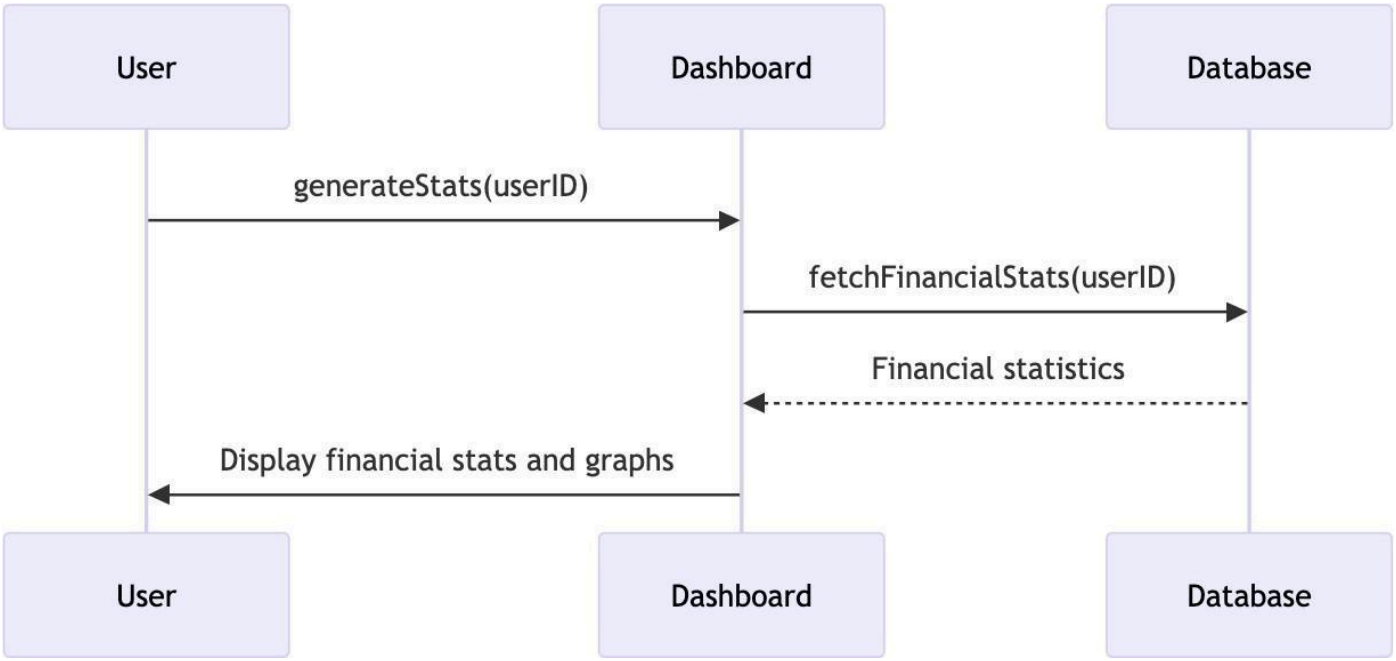
WHATSAPP INTEGRATION SEQUENCE DIAGRAM



E WAYBILL GENERATION SEQUENCE DIAGRAM



DASHBOARD SEQUENCE DIAGRAM



Design Rationale

Design Issues and Rationale

1. User Authentication and Registration Issue:

How should user authentication and registration be implemented to ensure security and scalability?

Alternatives Considered:

1. **Basic Email/Password Authentication**:
    a. Simple to implement.
    b. Vulnerable to security risks like brute force attacks.
2. **OAuth-Based Authentication (e.g., Google, Facebook)**:
    a. Enhanced security and user convenience.
    b. Adds dependency on third-party services.
3. **Two-Factor Authentication (2FA)**:
    a. Provides an additional layer of security.
    b. May complicate the user experience.

Chosen Solution:

- **Basic Email/Password Authentication with Password Hashing**:
    - Implemented password hashing (e.g., bcrypt) to secure user credentials.
    - Chosen for its simplicity and ease of implementation, given the project's scope and timeline.

Rationale:

- The project prioritizes a quick and straightforward implementation.
- Password hashing ensures sufficient security for user data.
- OAuth and 2FA were rejected due to added complexity and third-party dependencies.

2. Document Processing and Classification Issue:

How should the system classify and extract data from uploaded documents?

Alternatives Considered:

1. **Rule-Based Classification**:
    a. Uses predefined rules to classify documents.
    b. Limited flexibility and scalability.
2. **Machine Learning-Based Classification (e.g., NanoNets API)**:
    a. Highly accurate and scalable.
    b. Requires integration with external APIs.
3. **Manual Classification by Users**:
    a. Users manually select document types.
    b. Prone to human error and inefficiency.

Chosen Solution:

- **Machine Learning-Based Classification (NanoNets API)**:
    - Integrated NanoNets API for automated document classification and data extraction.

    - Provides high accuracy and scalability.

Rationale:

- Rule-based classification was rejected due to its inflexibility.
- Manual classification was rejected due to inefficiency and error-prone nature.

- NanoNets API was chosen for its accuracy and ability to handle large volumes of documents.

3. WhatsApp Integration for Document Upload Issue:
How should the system handle document uploads via WhatsApp?
Alternatives Considered:

1. **Direct File Upload via WhatsApp API**:
    a. Users upload files directly through WhatsApp.
    b. Requires robust file handling and validation.
2. **Email-Based Upload with WhatsApp Notifications**:
    a. Users upload files via email and receive notifications on WhatsApp.
    b. Adds an extra step for users.
3. **Cloud Storage Link Sharing**:
    a. Users share links to files stored in cloud storage (e.g., Google Drive).
    b. Requires users to have cloud storage accounts.

Chosen Solution:

- **Direct File Upload via WhatsApp API**:
    - Integrated Twilio API for WhatsApp to allow direct file uploads.
    - Implemented file validation and processing logic.

Rationale:

- Email-based upload and cloud storage link sharing were rejected due to added complexity for users.
- Direct file upload via WhatsApp was chosen for its simplicity and seamless user experience.

4. Dashboard Design and Data Visualization Issue:
How should the dashboard display financial statistics and graphs?
Alternatives Considered:

1. **Static Dashboard with Predefined Graphs**:
    a. Displays fixed graphs and statistics.
    b. Limited flexibility for users.
2. **Interactive Dashboard with Customizable Views**:
    a. Allows users to customize graphs and statistics.
    b. Requires more development effort.
3. **Third-Party Dashboard Tools (e.g., Tableau, Power BI)**:
    a. Provides advanced visualization features.
    b. Adds dependency on external tools.

Chosen Solution:

- **Interactive Dashboard with Customizable Views**:
    - Built using a frontend framework (e.g., React) and charting library (e.g., Chart.js).
    - Allows users to customize views and filter data.

Rationale:

- Static dashboards were rejected due to limited user flexibility.
- Third-party tools were rejected due to added cost and dependency.
- An interactive dashboard was chosen for its balance of flexibility and development effort.

5. E-Waybill

Generation Issue:
How should the system handle e-waybill generation?

Alternatives Considered:

1. **Manual E-Waybill Generation**:
    a. Users manually enter details to generate e-waybills.
    b. Prone to errors and time-consuming.
2. **Automated E-Waybill Generation with Prefilled Data**:
    a. Automatically fills data from user profiles and past filings.
    b. Requires integration with tax filing data.
3. **Third-Party E-Waybill Services**:
    a. Uses external services for e-waybill generation.
    b. Adds dependency and cost.

Chosen Solution:

- **Manual E-Waybill Generation**:
    - Users manually enter details to generate e-waybills.
    - Implemented a user-friendly form for data entry.

Rationale:

- Automated generation was rejected due to the complexity of integrating with tax filing data.
- Third-party services were rejected due to added cost and dependency.
- Manual generation was chosen for its simplicity and alignment with the project's scope and timeline.

Summary of Trade-Offs

Feature	Chosen Solution	Rejected Alternatives	Reason for Choice
User Authentication	Email/Password with Hashing	OAuth, 2FA	Simplicity and security
Document Processing	NanoNets API	Rule-Based,	Accuracy and scalability

		Manual Classification	
WhatsApp Document Upload	Direct File Upload via WhatsApp API	Email-Based, Cloud Storage Links	Seamless user experience

Dashboard Design	Interactive Dashboard	Static Dashboard, Third-Party Tools	Flexibility and customization
E-Waybill Generation	Manual Generation	Automated Generation, Third-Party Services	Simplicity and alignment with scope

Design Updates After Mid-Submission Evaluation

Platform Transition: Web to Mobile Application

Migration Strategy

- All web app functionalities have been successfully converted to a mobile application format

- Preserved core feature parity while optimizing for mobile user experience
- Implemented responsive design patterns to ensure compatibility across various device sizes

Technical Implementation

- **Mobile Framework**: Native Android development with React Native
- **UI/UX Adaptations**:
    - Redesigned dashboard layout for mobile screens
    - Converted web forms to mobile-friendly input interfaces
    - Implemented touch-optimized controls and navigation patterns

Integration Points

- Maintained all existing API connections and data processing workflows
- Optimized document upload process for mobile device camera integration
- Retained E-Waybill generation capability with mobile-friendly form inputs

Enhanced WhatsApp Integration

User Authentication Flow

- **New Feature**: WhatsApp-based login using mobile number verification
- **Process Flow**:
    1. User initiates login via WhatsApp
    2. User verification completes authentication

Data Segregation Architecture

- Implemented identifier-based data segregation using mobile numbers
- Data now stored with user-specific mobile number identifiers instead of common database
- Enhanced privacy and data security with user-specific data isolation

Class Modifications

Advanced Chatbot Capabilities

RAG-Based Support Implementation

- Implemented Retrieval-Augmented Generation (RAG) architecture for the support chatbot
- Enhanced response accuracy with domain-specific knowledge base integration
- Added context-aware responses through document retrieval mechanisms

Conversation Memory Architecture

- **Feature**: Chatbot now retains conversation history until explicitly closed
- **Implementation**:
    - Session-based conversation management
    - Context preservation across multiple user queries
    - Historical query reference for improved response relevance

Updated Sequence Diagram: WhatsApp Authentication Flow

User -> WhatsAppIntegration: Initiates login request
WhatsAppIntegration -> User: Requests mobile number
User -> WhatsAppIntegration: Provides mobile number
WhatsAppIntegration -> User: Sends OTP verification
User -> WhatsAppIntegration: Submits OTP
WhatsAppIntegration -> User: Verifies identity
WhatsAppIntegration -> Database: Associates WhatsApp ID with user account
WhatsAppIntegration -> User: Confirms successful login

Design Rationale for Updates

Mobile Application Transition

- **Issue**: How to maintain feature parity while optimizing for mobile experience?
- **Alternatives Considered**:
    1. Progressive Web App (PWA)
        - Pros: Easier transition from web
        - Cons: Limited access to native device features
    2. Native Mobile App
        - Pros: Full access to device capabilities
        - Cons: Requires separate development for iOS/Android

3. Hybrid App Framework
    - Pros: Single codebase for multiple platforms
    - Cons: Possible performance limitations
- **Chosen Solution**: Hybrid App Framework

WhatsApp Authentication

- **Issue**: How to securely implement WhatsApp-based login?
- **Alternatives Considered**:
    1. WhatsApp Business API direct integration
        - Pros: Official integration channel
        - Cons: Complex approval process and higher costs
    2. Third-party WhatsApp authentication service
        - Pros: Faster implementation
        - Cons: Additional dependency and security concerns
    3. Custom verification via WhatsApp
        - Pros: Greater control over authentication flow
        - Cons: Requires robust error handling
- **Chosen Solution**: Custom Verification

RAG-Based Chatbot

- **Issue**: How to improve chatbot response quality and context awareness?
- **Alternatives Considered**:
    1. Simple rule-based chatbot with expanded rules
        - Pros: Lower computational requirements
        - Cons: Limited response quality
    2. Full LLM implementation without retrieval
        - Pros: Natural language capabilities
        - Cons: Lacks specific domain knowledge
    3. RAG architecture with session memory
        - Pros: Combines domain knowledge with conversation context
        - Cons: Higher implementation complexity
- **Chosen Solution**: RAG architecture with session memory for its optimal balance of domain knowledge integration and conversational context awareness