Projects BTECH 2 CSE 2025

Supervisor: Dr TABUEU Fotso Laurent

Topic 1: Document Tracking System Specification

1. Project Overview

The system automates document tracking and management by processing scanned documents, extracting key data using OCR and AI, and converting them into Word format for further review and processing. Notifications alert users about deadlines, while real-time collaboration facilitates document management.

2. Actors

- **User (Client)**: Submits physical documents to the organization for processing.
- **Secretary**: Receives, scans, and uploads documents to the system.
- **Deputy Director**: Assigns scanned documents to the appropriate department for processing.
- **Processing Staff**: Processes the document and updates the status in the system within a predefined time.
- **AI System**: Automatically extracts key data from documents and triggers notifications about deadlines and document status.

3. Use Cases

1. **Submit Document** (User, Secretary)
2. **Scan Document** (Secretary)
3. **Extract Information** (AI System)
4. **Assign Document** (Deputy Director)
5. **Process Document** (Processing Staff)
6. **Monitor Deadlines** (AI System)
7. **Send Notifications** (System)
8. **Review and Validate** (Deputy Director, Processing Staff)

4. Detailed Entity Descriptions and Cardinalities

Document

Attributes:
- document_id: Unique identifier for the document.
- submission_date: The date when the document is scanned.
- status: Indicates whether the document is 'Pending', 'Processing', or 'Completed'.
- reference_number: Automatically generated upon document creation.

Relationships:
- A Document is submitted by one User (1:1).

- A Document can be assigned to many Processing Staff for handling (1:n).
- A Document is processed by the AI System for data extraction (1:1).

User
Attributes:
- user_id: Unique identifier for each user.
- name: Full name of the user.
- email: Contact email.
- phone: Phone number for contact.
Relationships:
- A User can submit many Documents (1:n).

Secretary
Attributes:
- secretary_id: Unique identifier for each secretary.
- name: Full name of the secretary.
- email: Contact email.
Relationships:
- A Secretary can handle many Documents (1:n).

Processing Staff
Attributes:
- staff_id: Unique identifier for each processing staff.
- name: Full name.
- department: The department to which the staff belongs.
Relationships:
- A Processing Staff can handle many Documents (1:n).

AI System
Attributes:
- ai_id: Unique identifier for the AI system process.
- extracted_data: Key information extracted by the AI (e.g., sender name, reference number, date).
Relationships:
- The AI System extracts data for many Documents (1:n).

Notification
Attributes:
- notification_id: Unique identifier for each notification.
- message: Content of the notification (e.g., 'Document pending processing').
- alert_type: Type of alert, such as deadline warnings.
Relationships:
- A Notification is triggered by the AI System for a Document (n:1).

Assignment (Association Class)
Attributes:
- assignment_id: Unique identifier.
- assignment_date: Date the document is assigned.
- deadline: Deadline for the processing staff to complete the task.
Relationships:

- Assignment connects Document and Processing Staff. A document can be assigned to one or more Processing Staff (1:n), and Processing Staff can have many Assignments (n:1).

5. Class Diagram with Associations

The updated class diagram includes entities like Document, User, Secretary, Processing Staff, AI System, Notification, and Assignment with relationships and cardinalities:
- Document ↔ User (1:n): A user can submit many documents.
- Document ↔ Secretary (1:n): A secretary handles many documents.
- Document ↔ AI System (1:1): The AI system processes a document for data extraction.
- Document ↔ Processing Staff (1:n): One or more staff members handle a document.
- Processing Staff ↔ Assignment (1:n): Each assignment links staff to a document.
- AI System ↔ Notification (1:n): The AI system triggers notifications.

Topic 2: Home Service Platform Specification

1. Project Overview

The Home Service Platform connects users with service providers for various at-home services, such as cleaning, plumbing, electrical work, and more. The platform facilitates booking, payment, and communication between users and service providers, while leveraging geolocation for optimized service delivery, path optimization for efficiency, and personalized recommendations for users.

2. Actors

- **User (Client)**: Requests services and makes payments through the platform.
- **Service Provider**: Offers services and manages job requests through the platform.
- **Admin**: Manages the platform, oversees service providers, and handles user queries.
- **AI System**: Provides recommendations and optimizations based on user behavior and preferences.

3. Use Cases

1. **Register User** (User)
2. **Register Service Provider** (Service Provider)
3. **Request Service** (User)
4. **Accept Job** (Service Provider)
5. **Geolocation Tracking** (User, Service Provider)
6. **Optimize Path** (AI System)
7. **Process Payment** (User, Service Provider)
8. **Send Notifications** (System)
9. **Review Service** (User)
10. **Get Recommendations** (AI System)

4. Detailed Entity Descriptions and Cardinalities

User

Attributes:

- user_id: Unique identifier for each user.
- name: Full name of the user.
- email: Contact email.
- phone: Phone number for contact.
- location: Current geolocation coordinates (latitude, longitude).

Relationships:

- A User can request many Services (1:n).

Service Provider

Attributes:

- provider_id: Unique identifier for each service provider.
- name: Full name of the provider.
- email: Contact email.
- phone: Phone number for contact.
- services_offered: List of services provided.
- location: Geolocation coordinates.

Relationships:

- A Service Provider can handle many Jobs (1:n).

Service

Attributes:

- service_id: Unique identifier for the service.
- type: Type of service (e.g., cleaning, plumbing).
- cost: Cost of the service.
- duration: Estimated duration for completion.

Job

Attributes:

- job_id: Unique identifier for the job.
- service_id: Associated service.
- user_id: User requesting the service.

- provider_id: Service provider assigned to the job.
- status: Job status (e.g., Pending, In Progress, Completed).
- scheduled_time: Time for the job.

Relationships:

- A Job is assigned to one Service Provider (1:1).
- A Job is requested by one User (1:1).

Payment

Attributes:

- payment_id: Unique identifier for the payment.
- job_id: Associated job.
- amount: Total amount charged.
- payment_method: Method of payment (e.g., credit card, PayPal).
- status: Payment status (e.g., Completed, Pending).

Relationships:

- A Payment is linked to one Job (1:1).

AI System

Attributes:

- ai_id: Unique identifier for the AI system process.
- recommendations: List of recommended services based on user preferences.
- optimized_path: The best route for service providers based on geolocation.

5. Class Diagram with Associations

- User ↔ Job (1:n): A user can request many jobs.
- Service Provider ↔ Job (1:n): A provider can handle many jobs.
- Job ↔ Payment (1:1): Each job has one associated payment.
- User ↔ Payment (1:n): A user can have multiple payments.

Security

- **Authentication**: JWT for securing endpoints.
- **Rate Limiting**: To ensure stability and prevent abuse.

Error Handling and Validation

- Validate inputs for all endpoints (e.g., email format, service availability).
- Standardized error responses with appropriate HTTP status codes.

Monitoring and Logging

- Use tools like Prometheus for monitoring API performance.
- Implement structured logging for debugging and auditing.

Topic 3: Monitored Search System for Large Spaces

Introduction

- **Context**: Searches in large spaces require speed and efficiency. Due to the constraints related to the installation of fixed sensors on newly arrived boats, an alternative solution is necessary.
- **Objectives**: Create a mobile and real-time monitored search system using animal equipped with GPS trackers and drones to map and identify suspicious areas. Everything will be centralized on an AI platform.

Features:

- Dashboard;
- User management;
- Management of maps;
- Management of boat and cargo information (names, types, capacities, etc.);
- Monitoring of searches;
- Management of search planning;
- Tracking and overlaying GPS positions or crossing GPS data;
- Live retransmission of the search on screens, allowing office teams to see the activity in real-time and communicate with the dog handlers as needed;
- Alerts and notifications;
- Report generation;
- Report validation by electronic signature (fingerprint, stylus);
- Mailing;
- Historical data;
- Related features:
- Application configuration settings;
- Login/logout;

Classes:

1. **User**
    - Attributes:
        - id: int
        - name: String
        - email: String
        - password: String
        - role: String
        - lastLogin: DateTime
    - Methods:
        - login()
        - logout()
        - changePassword()
2. **Drone**
    - Attributes:
        - id: int
        - model: String
        - serialNumber: String
        - status: String
        - gpsPosition: String
    - Methods:
        - getPosition()
        - track()
        - start()
3. **Sensor**
    - Attributes:
        - id: int
        - type: String
        - value: String
        - dateRecorded: DateTime
    - Methods:
        - recordData()
4. **IoT**
    - Attributes:
        - id: int
        - name: String
        - status: String

        - position: String
    - Methods:
        - updateStatus()
5. **Boat**
    - Attributes:
        - id: int
        - name: String
        - type: String
        - capacity: int
    - Methods:
        - getDetails()
6. **Cargo**
    - Attributes:
        - id: int
        - description: String
        - weight: float
        - destination: String
        - boat: Boat
    - Methods:
        - trackCargo()
7. **Search**
    - Attributes:
        - id: int
        - date: DateTime
        - location: String
        - duration: float
        - responsible: User
        - drones: List<Drone>
        - status: String
        - dogs: List<Dog> (newly added attribute)
    - Methods:
        - planSearch()
        - trackProgress()
        - closeSearch()
        - addDog() (new method)
8. **Report**
    - Attributes:

        - id: int
        - search: Search
        - date: DateTime
        - content: String
        - electronicSignature: String
    - Methods:
        - generateReport()
        - sign()

9. **Notification**
    - Attributes:
        - id: int
        - message: String
        - date: DateTime
        - recipient: User
    - Methods:
        - sendNotification()

10. **History**

- Attributes:
    - id: int
    - action: String
    - user: User
    - date: DateTime
- Methods:
    - addAction()

11. **GPS**

- Attributes:
    - id: int
    - latitude: String
    - longitude: String
    - drone: Drone
    - search: Search
- Methods:
    - crossData()

12. **Parameters**

- Attributes:
    - id: int

    - name: String
    - value: String
- Methods:
    - update()

13.**Mailing**

- Attributes:
    - id: int
    - recipient: User
    - message: String
- Methods:
    - sendEmail()

14.**Dog (new class)**

- Attributes:
    - id: int
    - name: String
    - breed: String
    - age: int
    - healthStatus: String
    - equipment: List<Equipment>
    - handler: Handler
    - searches: List<Search>
    - medicalVisits: List<MedicalVisit>
- Methods:
    - addEquipment()
    - planMedicalVisit()
    - associateSearch()

15.**Equipment**

- Attributes:
    - id: int
    - description: String
    - type: String
- Methods:
    - checkEquipment()

16.**Handler**

- Attributes:

    - id: int
    - name: String
    - experience: int
    - dogs: List<Dog>
- Methods:
    - assignDog()

17. **MedicalVisit**

- Attributes:
    - id: int
    - date: DateTime
    - veterinarian: String
    - results: String
    - dog: Dog
- Methods:
    - scheduleVisit()
    - addResults()

Search Attributes:

- Date of the search
- Start and end time of the search
- Total duration
- Areas searched
- Total area searched
- Percentage searched
- List of found objects (Relation with FoundObject)
- List of incidents (Relation with Incident)
- Problems encountered
- Results of the search
- Observations (List of observations made during the search)
- Methods:
    - planSearch()
    - trackProgress()
    - closeSearch()
    - addFoundObject(FoundObject)
    - addIncident(Incident)

    - addObservation()

FoundObject

- Attributes:
    - id: int
    - description: String
    - type: String (e.g., weapon, drug, etc.)
    - value: String (quantity, weight, etc.)
    - GPS Coordinates: Latitude, Longitude, Altitude
    - Search: Relation to the search where the object was found
- Methods:
    - recordObject()
    - associateGPSCoordinates()

Incident

- Attributes:
    - id: int
    - type: String (e.g., mechanical, environmental, etc.)
    - description: String
    - GPS Coordinates: Latitude, Longitude, Altitude
    - Search: Relation to the search where the incident occurred
- Methods:
    - recordIncident()
    - associateGPSCoordinates()

GPS

- Attributes:
    - id: int
    - latitude: String
    - longitude: String
    - altitude: String
    - FoundObject: Optional relation to the found object
    - Incident: Optional relation to the incident
- Methods:
    - recordCoordinates()
    - associateObjectOrIncident()

Observation

- Attributes:

        - id: int
        - description: String
        - search: Relation to the search
- Methods:
    - addObservation()

Updated Relationships:

1. Search ↔ FoundObject: 1 ↔ 0..*
2. Search ↔ Incident: 1 ↔ 0..*
3. FoundObject ↔ GPS: 1 ↔ 1
4. Incident ↔ GPS: 1 ↔ 1
5. Search ↔ Observation: 1 ↔ 0..*

Associations and Cardinalities:

1. Handler ↔ Dog
    - Cardinality: 1 ↔ 0..*
    - A handler can have multiple dogs, but each dog is under the responsibility of a single handler.
2. Dog ↔ Search
    - Cardinality: 0..* ↔ 0..*
    - A dog can participate in multiple searches, and a search can involve multiple dogs.
3. Dog ↔ Equipment
    - Cardinality: 1 ↔ 0..*
    - A dog can have multiple associated equipment, but each equipment belongs to a single dog.
4. Dog ↔ MedicalVisit
    - Cardinality: 1 ↔ 0..*
    - A dog can have multiple medical visits, and each medical visit is associated with a dog.
5. Search ↔ Drone
    - Cardinality: 0..* ↔ 0..*
    - A search can use multiple drones, and a drone can participate in multiple searches.
6. Search ↔ User (Responsible)
    - Cardinality: 1 ↔ 1
    - A search is conducted by a single responsible user of the system.
7. Search ↔ Report
    - Cardinality: 1 ↔ 1
    - Each search is associated with a unique report.
8. User ↔ History

- Cardinality: 1 ↔ 0..*
- Each user has a history of actions, and each action is linked to a user.

9. Search ↔ GPS
    - Cardinality: 1 ↔ 1
    - Each search has an associated GPS position.

Use Cases:

1. **Boat Search:**

   Description:
   A canine operator (or another member of the search team) logs into the application via an authentication system. Once logged in, they can initiate a search for a given vessel, record incidents and found objects. Each object and incident is geolocated with GPS coordinates.

   Steps:

   1. Authentication: The canine operator logs into the application.
   2. Start the search: The operator selects a vessel, the areas to search, and starts the search.
   3. Recording incidents and found objects: During the search, the operator can add incidents or found objects by specifying GPS coordinates and other details.
   4. Close the search: The operator concludes the search by generating a preliminary report.

2. **Management of Search Reports:**

   Description:
   After the search, a search report is generated, validated, and shared with stakeholders (ship captain, supervisor, etc.). The system allows adding observations, signing the report, and archiving this data.

   Steps:

   1. Authentication: The supervisor logs into the system.
   2. View reports: The supervisor accesses the list of search reports.
   3. Validation and signature: The supervisor validates and signs the report.
   4. Sharing and archiving: The report is shared with stakeholders (via email or download).

Mockups of the Frontend Interface:

1. **Login Page**
    - Contains input fields for email and password.
    - Button "Log In."
    - Error message if the credentials are incorrect.
2. **Search Start Interface**
    - Dropdown list to select the vessel to search.
    - Selection of areas to search (checkboxes).

    - Button to start the search.
3. **Incident/Object Addition Interface**
    - Form to add an incident or a found object:
        - GPS Coordinates (latitude, longitude, altitude).
        - Description of the incident or object.
    - Button "Add."
4. **Search Reports Dashboard**
    - List of generated reports.
    - Button to view, validate, and sign a report.

Dashboard Components:

- **Navigation Bar**:
    - Access to different sections (Ongoing Searches, Reports, Incident/Object Management).
    - Link to settings and logout.
- **Map of Ongoing Operations**:
    - Visualization of ongoing searches with GPS position.
    - Icons for each active search.
- **Quick Action List**:
    - Start a new search.
    - Access reports pending validation.
    - Add an incident or object.

Mockup of the Dashboard:

```
+-------------------------------------------------------------+
|                        Dashboard                             |
+-------------------------------------------------------------+
| [Nav] Searches | Reports | Incidents/Objects | Settings    |
+-------------------------------------------------------------+
|                  Ongoing Operations Map                     |
| [Map with GPS position of ongoing searches...]              |
|                                                             |
+-------------------------------------------------------------+
| Actions Quick:                                              |
| [Start a new search] [Add an incident]                      |
| [Add an object] [View pending reports]                      |
+-------------------------------------------------------------+
```

"Start a New Search" Interface Integrated into the Dashboard

Description:
This interface will allow the user to initiate a new search directly from the dashboard.

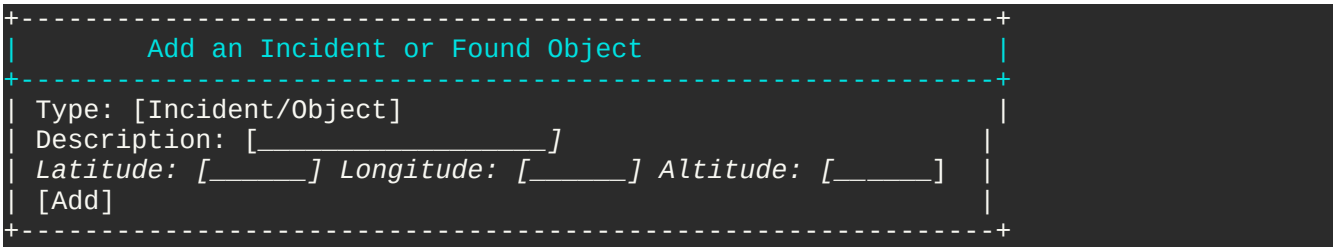
Mockup:

```
+-------------------------------------------------------------+
```

Incident and Object Management (Dashboard Section)

Description:
In this section, the user can add incidents or objects found during the search. Everything is centralized in the dashboard for better accessibility.

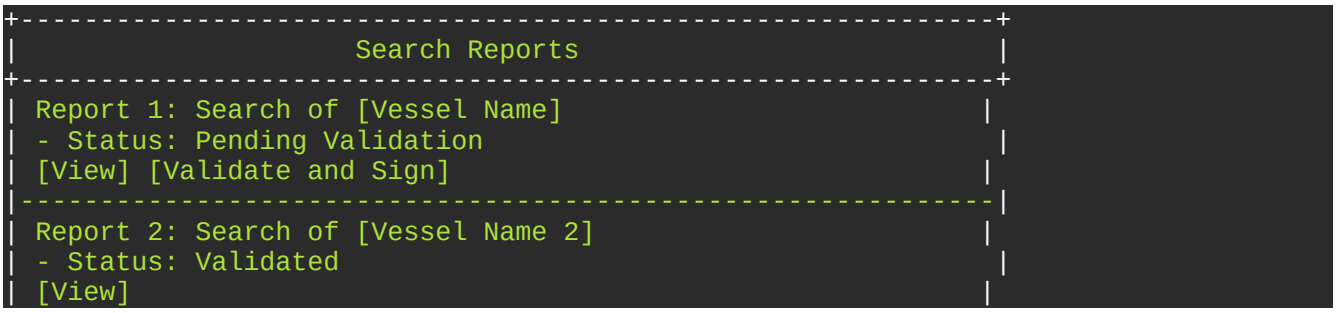
Mockup:



Search Reports

Description:
The user can access reports from the dashboard, review them, validate, or add observations.

Mockup:



Below is a high-level API design for the application, oriented towards RESTful principles, which can later be implemented using Django Rest Framework. The API design includes endpoints for the main entities described in your document.

Topic : Food Distribution Application

Comparison of Food Platforms

This paper presents a comparison of the different platforms used in the field of food distribution. The objective is to identify the features, advantages and disadvantages of each solution.

Application Name	Main features	Benefits	Disadvantages
FoodBank Manager	Food bank management (stocks, requests, distributions)	Robust features, advanced management tools	May lack tools for more dynamic management
Meal Tracker	Meal tracking and food intake management	Good management of individual contributions	Limited distribution features
Nourish Now	Recovery and redistribution of food surpluses	Strong partner network, connection between donors and beneficiaries	User interface can be improved
Food Rescue	Connecting restaurants and charities	Ideal for restaurants looking to reduce their surplus	Less suitable for long-term inventory management
Share Food	Collaborative platform for sharing food between individuals and organizations	Promotes local solidarity	Lack of professional features for rigorous management

Analysis of Results

• FoodBank Manager: Offers robust features for inventory management, but may lack tools for more dynamic management.
• Meal Tracker: Excellent for tracking intake, but has limited dispensing features.
• Nourish Now: Very effective for food recovery, with a good network of partners, but may need improvements in the user interface.
• Food Rescue: Ideal for restaurants, although not suitable for long-term food management.
• Share Food: Promotes solidarity, but lacks professional features for more rigorous management.

Functional Requirements Plan

This document presents the functional requirements plan for the food distribution application.

ID	Functional Requirements	Brief Description
BF-01	Registration and Authentication	Allows beneficiaries, volunteers and administrators to create and secure an account via email or phone number with OTP.
BF-02	User Profiles	Management of beneficiary, volunteer and administrator profiles with

		information specific to each role.
BF-03	Stock Recording	Allows NGOs and partners to add food items with details on type, quantity and expiry date.
BF-04	Real-Time Inventory Tracking	Visualize stock levels and receive notifications of shortages.
BF-05	Supplier Management	Interface allowing suppliers to record donations and track their deliveries.
BF-06	Request for Help	Beneficiaries submit a request for food based on their specific needs.
BF-07	Attribution and Validation	The application automatically assigns requests based on stock availability and a valid administrator before confirmation.
BF-08	Distribution Planning	Creation and management of collection points and routes for optimized distribution.
BF-09	Real Time Tracking	Track distributions by beneficiaries and volunteers via the application.
BF-10	Monetary Donations	Possibility to make donations online via bank card, mobile money or bank transfer.
BF-11	Food Donation Management	Interface for proposing and organizing the collection of in-kind donations.
BF-12	Integrated Messaging	Direct communication between beneficiaries, volunteers and administrators via internal messaging.
BF-13	Notifications	Alert system for distribution recalls, request confirmation and stock shortages.
BF-14	Online Assistance	User support via chatbot and customer service to answer questions.

Specification Management for a Food Distribution Application

1. Project Context

Objective:
To develop a mobile and web application enabling the efficient distribution of food supplies to populations in need, in collaboration with NGOs, volunteers and suppliers.

2. Functional Requirements

2.1. User Management

Registration and Authentication:
- Beneficiaries, volunteers and administrators can create an account via email or phone number.
- Secure authentication with password and OTP verification.
User Profiles:
- Beneficiaries: Provide information about their situation (number of dependents, specific needs).
- Volunteers: Indicate their availability and intervention area.
- Administrators: Manage distributions, check beneficiaries and stocks.

2.2. Inventory and Supplier Management

Stock Recording:
- NGOs and partners can record available food with details (type, quantity, expiry date).
Real-Time Stock Monitoring:
- Visualization of stock levels and alerts in case of shortage.
Suppliers:
- Interface to record donations and manage deliveries.

2.3. Distribution of Food

Request for Aid:
- Beneficiaries can request food according to their needs.
Assignment and Validation:
- The application automatically assigns requests according to stock availability.
- Validation by an administrator before confirmation.
Distribution Planning:
- Creation of collection points and management of routes for deliveries.
Real-Time Tracking:
- Beneficiaries and volunteers can track the status of the distribution.

2.4. Payment and Management of Donations

Monetary Donations:
- Payments via bank card, mobile money and bank transfer.
Food Donation Management:
- Donors can offer food and organize their collection.

2.5. Communication and Support

Integrated Messaging:
- Beneficiaries and volunteers can communicate with administrators.

Notifications:
- Distribution reminders, request confirmation and stock alerts.
Online Support:
- Chatbot and customer service to answer questions.

3. Non-Functional Requirements

3.1. Security
- Encryption of user data.
- Protection against fraud and abuse.
3.2. Performance
- Ability to handle 10,000 simultaneous users.
- Pages load in less than 2 seconds.
3.3. Accessibility
- Interface adapted for people with disabilities.
- Compatible with mobile browsers and Android/iOS applications.
3.4. Compatibility
- Works on multiple devices (smartphones, tablets, computers).
- Multi-language support (French, English, local languages).

4. Documentation of Requirements

Functional Requirements Specification
Title: Food Distribution Application Functional Requirements Specification
Version: 1.0
Date: [Insert date]
Prepared by: [Your name]

5. Project Management

Methodology
- Use of the Agile method for iterative development.
- Breaking down the project into sprints of 2 to 4 weeks.
Stakeholder Engagement
- Beneficiaries: Feedback to improve the application.
- NGOs and Suppliers: Consultation for the optimization of distribution flows.
Testing
- Development of unit and functional tests to validate functionalities.
- Load testing to ensure system scalability.

6. Change Management Plan

Change Request Form:
- Allow stakeholders to submit suggestions.
Impact Analysis:
- Study the consequences of the modifications on the project.
Approval Process:
- Validation of changes before implementation.

Conclusion

The implementation of this application would optimize the management of food distribution while ensuring increased transparency and efficiency. By following this methodical approach, the development team can ensure a project aligned with the needs of beneficiaries and partners.

Members

Full name	Email	Github username	Topic	Group ID