TED UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

CMPE492 - LOW LEVEL DESIGN REPORT

Project Title

ALSApp (Agriculture and Livestock Support Application)

Project Supervisor

Venera ADANOVA

Project Juries

Ayşe Yasemin Seydim

Tansel Dökeroğlu

Project Members

Engin Samet Dede

Sevgi Dilay Demirci

Gizem Ünsal

09/03/2025

Table Of Contents

1. Introduction

This report describes the low-level architecture and the design of ALSApp (Agriculture and Livestock Support Application) by providing an in-depth technical specification for the software's implementation. In this report, the previous High-Level Design report is transitioned into detailed class structures, object interactions, APIs, and database schemas to ensure a modular, scalable, and maintainable system.

1.1 Project Description

The goal of our project is to develop an application that will make tracking easier for farmers in a variety of ways. Furthermore, this platform combines all the necessary resources to improve livestock performance and farming activities. Users will be able to enter their animals' health data and receive relevant information, track their crops digitally for optimal efficiency, and track the growth and development of their crops and animals in real time using an easy-to-use interface that is favourable to both farmers and livestock. By consolidating all agricultural and livestock management needs onto a single platform, this system enables users to manage their operations in a more organized, efficient, and planned manner. It also provides instant updates on industry developments, helping users stay informed and responsive to industry changes. They will be aware of government support, mainly offered by the Ministry of Agriculture and Forestry, and benefit from funding opportunities. In addition, the following process can also be used by the Ministry of Agriculture and Forestry staff to keep track of the reliability and accuracy of support.

1.2 Purpose of the system

The ALSApp seeks to create an integrated digital platform that is built for the farmer and livestock breeder to simplify the management & monitoring of their farm and livestock operations. The application brings all the defined key features into a single application like crop and livestock tracking, real time weather update and easy access to the government support programs.

The system enables users to develop digital literacy and address the issues of digital resource operational inefficiencies by providing intuitive tools. Farmers can get notified if something is wrong with their crops and livestock, monitor, get the latest government support information and so forth. Additionally, Ministry of Agriculture and Forestry staff can also monitor the reliability and accuracy of the support delivered via the platform.

Through the ALSApp we strive to bridge the gap between modern digital solutions and traditional farming practices to increase productivity, sustainability and increase awareness of governmental assistance programs. This system design, however, telegraphs that it's easy to use, scalable, and secure for farmers as well as ministry officials, providing a robust solution.

1.3 Object Design Trade-offs

1.3.1 Usability vs. Functionality

The system is designed to be easy to use by farmers and livestock breeders with differing levels of digital literacy. Users won't have a large learning curve when it comes to using the interface. One of the important goals is to include crop and livestock tracking, real time weather updates and easy one stop shop of government support for the functionality. These features are designed to address directly the special requirements of the agricultural and livestock sectors. However implementations of functionalities that will lead to complex interfaces will be avoided since the main goal is to have a user-friendly UI.

1.3.2 Efficiency vs. Accuracy

Efficiency is an important focus of the app since farmers and livestock breeders will rely on real-time weather updates, livestock health tracking, and government support notifications. The system prioritizes fast data retrieval and minimal processing delays. Furthermore, accuracy of the system is crucial as it can affect the yield and the animals' health. Thus, weather predictions and analytics should be updated regularly and strict data validation should be made for crop and livestock tracking to ensure correct health status and harvest schedules. So, the accuracy and the efficiency should be balanced.

1.3.3 Security vs. Usability

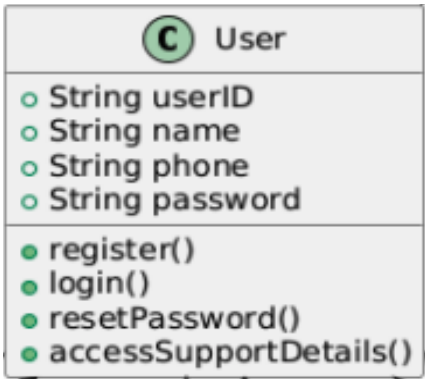
The security features of ALSApp play a critical role since the application manages delicate agricultural, livestock and financial support information. However, in real life, being more secure may require sacrificing some conveniences which are not wanted for this case, thus usability is prioritized. To balance both elements, users will encounter an optimal security experience through security prompts that are easy to understand. The system displays straightforward security alerts throughout the UI as simple notifications rather than technical warning messages.

1.3.4 Integration vs. Complexity

The app will work with external systems such as real time weather services and the Ministry of Agriculture and Forestry database. The availability of accurate and up to date information will be guaranteed with utilizing integration. However, adding multiple integrations can increase the complexity of the system. This might lead to potential maintenance challenges and performance bottlenecks. To avoid this API versioning will be used so that when new features are introduced or API structures change, older versions will still remain functional for existing users or integrations. Also modular adapters will be used so that the core system can stay the same when the app connects with multiple external services.

1.4 Interface documentation guidelines

Throughout the report, all classes, attributes and methods will be named in the camel case format to maintain consistency. The class names will begin with a capital letter and the class interface descriptions will be given similar to the following format:



Class: ClassName Class (User Class)

Attributes: typeOfAttribute nameOfAttribute (String userID)

Functions: methodName(parameters)(register())

1.5 Engineering standards

1.5.1 UML (Unified Modeling Language) and IEEE Software Design Standards

The UML guidelines will be used throughout this report for the class interfaces, diagrams, scenarios and subsystem compositions. To represent object-oriented relationships class diagrams and to define interactions between systems, sequence diagrams will be used. We will make use of the Use Case Diagrams for defining user interactions and the State Diagrams for system behavior representation. To generate all these diagrams, UML will be a helpful and easy to use language. The Low-Level report will be following IEEE's standards for any citation. Also, the IEEE guidelines will be followed to ensure structured, well-documented, and maintainable software development.

1.5.2 ACM Code of Ethics and Professional Conduct

The ACM Code is designed to inspire and guide the ethical conduct of all computing professionals, including current and aspiring practitioners, instructors, students, influencers, and anyone who uses computing technology in an impactful way. Additionally, the Code serves as a basis for remediation when violations occur[6]. ALSApp will adhere to this code to ensure that ethical principles guide its design, implementation, and usage. Developers will be responsible for ensuring that the app is reliable, secure, and beneficial to its users. Furthermore, All user data

containing the farm records, government support applications and weather tracking will be protected using encryption, secure APIs, and strict authentication.

1.6 Definitions, acronyms, and abbreviations

1.6.1 Definitions

ALSApp: Agriculture and Livestock Support Application, a digital platform for farmers and livestock breeders.

Crop: Cultivated plants grown on a large scale for food, fiber, or other commercial purposes. Examples include wheat, corn, and rice.

Crop Tracking: The process of monitoring and managing crop growth, health, and yield using digital tools.

Compatibility: The application's ability to work seamlessly with different devices, platforms, or systems without conflict.

Humidity: A quantity representing the amount of water vapor in the atmosphere or in a gas.

Factual Errors: Mistakes or inaccuracies in the data or information provided by the system that can impact its reliability or decision-making.

Yield: The amount of produce harvested from a crop or the output from livestock, often measured per area or animal.

1.6.2 Acronyms and Abbreviations

API: Application Programming Interface

UI: User Interface

UML: Unified Modeling Language

2. Packages

In this section, the packages used to create the ALSApp application are added as a diagram. The first diagram below represents ALSApp as a whole. This client server architecture is shown in this package diagram of Alsapp where it is layered having two domains, one for client side and another for server side. The user interacts with Widget and View on the Client side, which renders interface and process user inputs. They then send this data or initiate this action to server-side components represented here as Server.Model, Server.Service, and Server.API. In essence, the Client makes calls to the Server, listens for responses, and updates the UI as a result.

The incoming requests are received on the Server side, passed on (if Controller is there) to the Service through the API. The Main logic (business rules) is handled by the Service rule retrieving or storing data in the Database based on rules from the Model. This is an approach that allows users of a separation of concerns: the Client only deals with the interaction and presentation aspects, while the Server is responsible for the logic application and persistence of data. Such an organization enhances the maintainability, scales well, and prevents the system from becoming complex to determine and evolve.

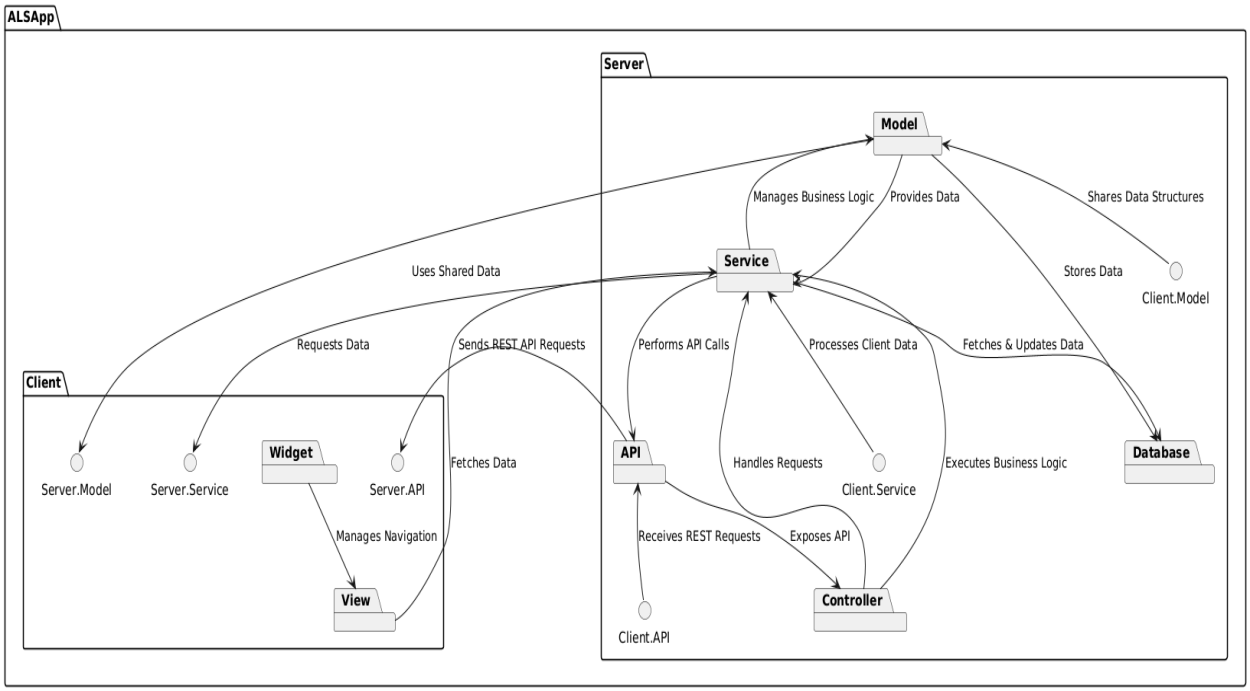


Figure 1: Alsapp Diagram

2.1 Server

2.1.1 Model

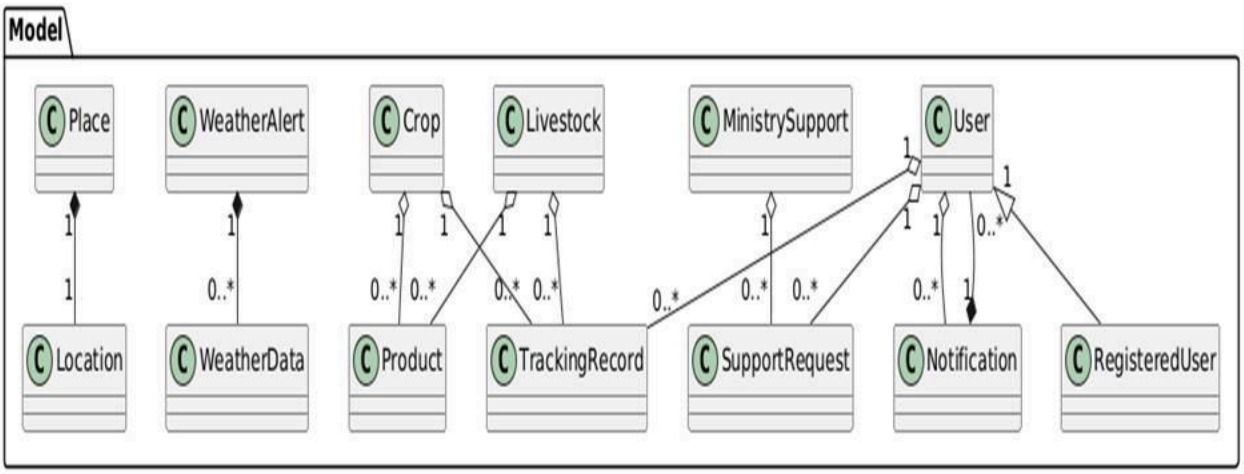


Figure 2: Model package

- **Place:** this stands for a geographical location, that is covering a region dedicated to the farmer or livestock breeder. Relates with weather alerts and crop and livestock tracking records entities. Has a one-to-many relationship with Location indicating that each place can encompass multiple specific locations.
- **Location:** has more detailed identifiers about the user's geographical information. Stores the smaller areas or region-specific data. Linked to Place to provide more detailed localization.It is connected to the Place to establish broader geographical context..
- **WeatherAlert:** This is providing notifications on weather situations that may negatively impact agricultural activity.
- **WeatherData:** Stores real time data about weather at certain locations such as calculating temperature, humidity etc.
- **Crop:** Includes information like variety, planting date, stage, and health among many farmed crop types. Attached to TrackingRecord to track history of a crop's growth and health so it can be Analyzed and Future Planned for users.

- **Livestock:** Manages information about different types of livestock species such as health, breeding information, productivity metrics etc. This is associated with TrackingRecord for continuous tracking and administration of productivity and health.
- **Product:** It represents crop and Livestock products like harvests and animal products. Outcome products linked with both Crop and Livestock.
- **TrackingRecord:** Detailed records about agricultural activities such as crop treatment and livestock vaccine logs are recorded in this section. Providing a detailed historical record from Crop and Livestock to which it is linked.
- **MinistrySupport:** Stores information about programs and supports applied by the government to assist the agricultural activities. Linked to SupportRequest to manage the applications and interactions with these support programs.
- **SupportRequest:** Individual applications or request for support of the users, and their status and result tracking. These were linked to MinistrySupport to associate the request with the particular programs that the request falls under.
- **User:** Basic profile of information for agriculture workers, administrators, and farmers. RegisteredUser is directly linked to Notification to process alerts and messages related to the user through Notification as a parent entity.
- **RegisteredUser:** The more detailed and authenticated profile of a user who has registered in the system and we can give more features and access to the user. It inherits from User; it has additional attributes, and there are additional specific data access rights.
- **Notification:** It handles notifications and alerts that are served to the users such as when there's a change in weather, it also deals with notifications and alerts for when there is an update in the support program and similar. User related, as it is related to User, to guarantee that the notifications are properly tailored and delivered to the appropriate user depending on its needs and actions.

2.1.2 Service Package

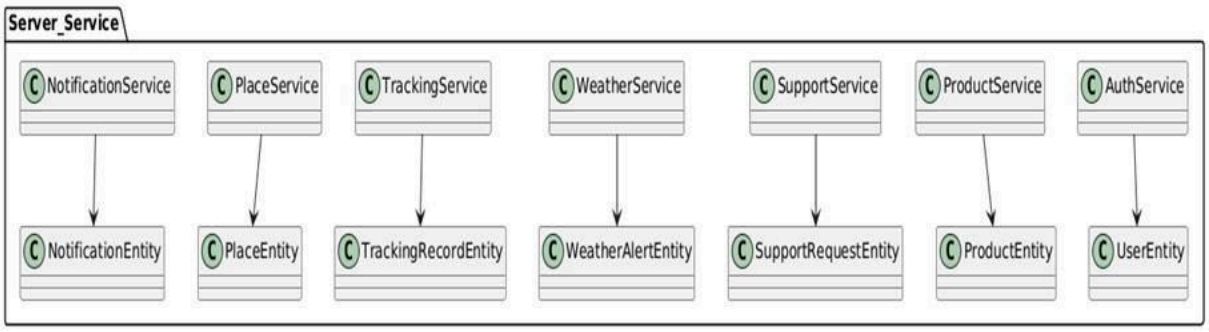


Figure 3: Service package

- **NotificationService:** Handles all aspects of notifications to the users within the app and ensures that users are informed about important events like harvest time, changes like weather alerts or support programs that the government has been provided. Integrates with weather service, support services and tracking service to gather actionable events. Associated with NotificationEntity to represent the data structure for store notification details (sender, message content and status).
- **PlaceService** Handles geographic and location-based information, supporting operations related to managing places within the application. Associated with PlaceEntity, Contains details about places, such as geographical coordinates, names, and related attributes.
- **TrackingService:** Manages tracking of agricultural activities.Monitors crop and livestock recording and organizes these records. Analyzes them and if needed generates alerts for users based on the anomalies on these recordings. Associated with TrackingRecordEntity, which stores records of tracking data-growth statistics, health metrics, and other related information related to agriculture or livestock.
- **WeatherService:** Fetches, processes, and delivers weather data from an external weather API to the application, then provides operations related to weather forecasting and alerting that are crucial for agricultural planning and operations about it. Retrieves current and forecasted weather data based on the user's location. Associated with WeatherAlertEntity, that holds information about weather alerts and conditions that are important for users-like storms and rainfalls-.
- **SupportService:** handles the management of support programs that are available to farmers and manages the processes and tracking of support claim, status updates and

outcomes. Fetches information about support from government databases or APIs associated with SupportRequestEntity, which manages individual processes of support requests, detailing user applications, status updates, and outcomes.

- **ProductService:** Deals with products of agricultural activity, harvested goods and other products obtained from livestock activity. Linked to ProductEntity that stores the information about agricultural products, which include type, quantity, metrics for quality and date of production.
- **AuthService:** All authentication related functionalities such as how the user logs in or registers and validating user credentials against stored data allowed for securing sessions by providing adequate security so that user data cannot be accessed without authorization. It's associated with UserEntity, containing the user-specific information essential to the system access and identity management, which includes username, password, etc.

2.1.3 Controller

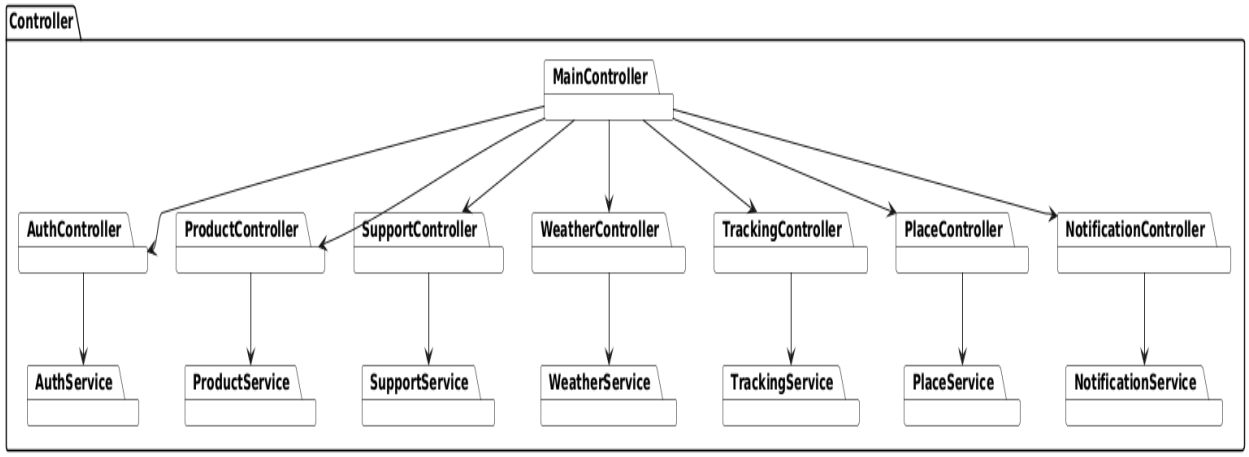


Figure 4: Controller package

- **MainController:** Central hub for routing requests and specialized controllers. High level flow of data and operation in the application. It can handle error handling, request logging, and authentication checks across the controllers.
- **AuthController:** the controller in charge of all features of the authentication and authorization so that all the users who access the application are verified to authorize them. It works with login, registration, session management and password recovery. Connected to **AuthService**
- **ProductController:** Functions relating to management of product listing are carried out by ProductController. It adds, updates and removes product data. Connected to **ProductService.**
- **SupportController:** Enables communication between support programs and farmers through its management system. It enables user applications for support programs while monitoring application statuses through integration with external systems that provide updates about the support programs. Connected to **SupportService.**
- **WeatherController:** delivers current weather data as well as hazard warning notifications to users. The application retrieves weather information regarding forecasts and severe weather alerts through external APIs to display it to users. Connected to **WeatherService.**

- **TrackingController:** Operates as the central system to monitor agricultural operations that include planting actions together with harvesting activities and livestock monitoring tasks. **TrackingService** allows the system to log and retrieve tracking data which it analyzes to produce alerts based on this information.
- **PlaceController:** Manages geographical and location-based information within the application. The application stores detailed place information through addition and update operations to improve location precision and provide adjusted weather warnings. Connected to **PlaceService**.
- **NotificationController:** Enables the delivery of application-based alerts to users regarding different system events and application updates. Through this component the system sends real-time notifications that follow user preferences and triggered events while managing all subscription settings. Connected to **NotificationService.**

2.1.4 API

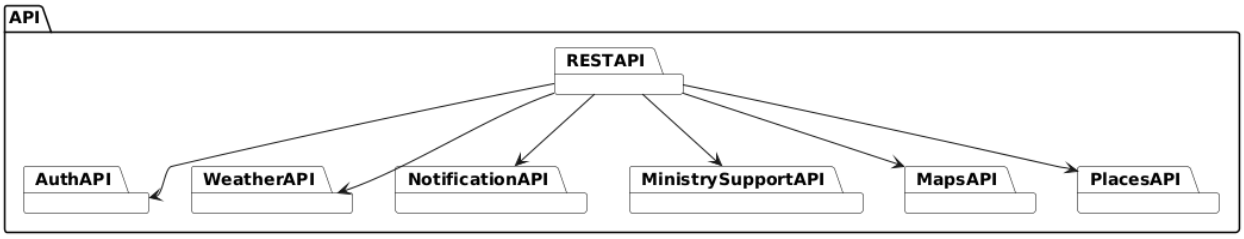


Figure 5: API package

The API layer of the ALSApp is meant to encapsulate the communication between a client side and server side operations by use of a centralized RESTAPI interface. This configuration is suitable for efficient and secure processing of data across various application services.

AuthAPI

Handles all authentication-related operations, including user login, registration, and session management.

WeatherAPI

Provides weather data crucial for agricultural planning.

NotificationAPI

The application delivers alerts through notifications which provide information about weather updates and important app system support details to all users.

MinistrySupportAPI

Handles specific governmental agricultural support details.

MapsAPI

The app uses integrated mapping services to enable location-based functions.

PlacesAPI

This API maintains thorough details regarding places and locations that matter to the users' agricultural farming activities.

RESTAPI

This API serves as the system controller that executes HTTP requests towards selected APIs according to the endpoints specified by users. The API ensures that requests get proper consistent handling and authentication verification and that responses maintain correct formats from all APIs.

2.1.5 Database

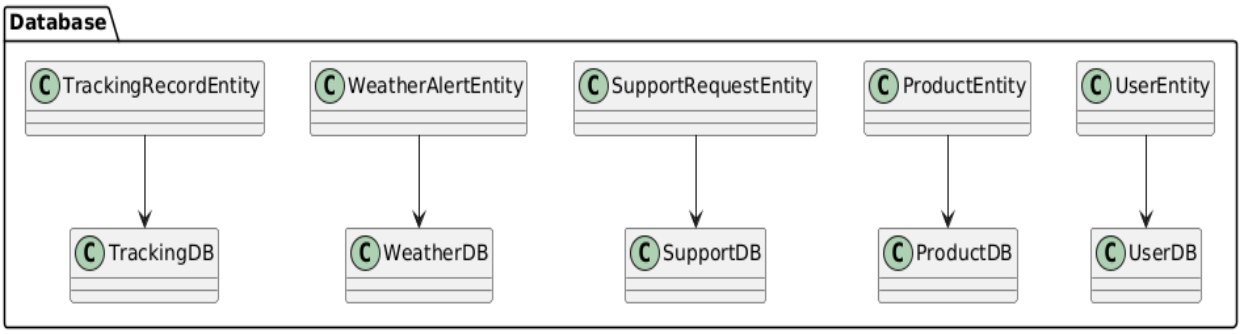


Figure 6: Database package

The purpose of database packages is to handle different kinds of essential data needed for application operations. Each data type maintains its own database because specialized management and optimized performance require separate domains for tracking, weather, support, products, and user management.

TrackingRecordEntity and **TrackingDB:**

- The agricultural tracking database TrackingDB maintains complete data records regarding activity tracking, including planting operations, growth stage surveillance, and animal healthcare activities.
- TrackingRecordEntity contains multiple fields that include date stamps as well as activity types alongside crop or livestock identification numbers, health measurements and treatment logs.
- The system allows users to track agricultural data and perform historical reviews which leads to improved decisions and more efficient farming practice execution.

WeatherAlertEntity and WeatherDB

- WeatherDB stores information on the weather condition and alerts, which are relevant for performing and responding to those agro-environmental factors

- The WeatherAlertEntity groups alert types with severity levels, location specifics, timestamp, and predictive data for publicising weather threats to the end users.
- In accordance, the system provides current and predicted weather data to the user in a form to minimize operational risks and improve agricultural management.

SupportRequestEntity and SupportDB

- SupportDB manages information related to applications and interactions with agricultural support programs.
- SupportRequestEntity records details such as user IDs, program IDs, application status, submission dates, and decision outcomes.
- Facilitates the management of government or institutional support programs, tracking user applications, and the distribution of benefits.

ProductEntity and ProductDB

- ProductDB contains detailed information on agricultural products derived from crops and livestock managed by the application.
- ProductEntity contains information about product types together with quantitative data, quality assessment results, harvest dates, and market data.
- The system helps manage inventory alongside quality inspections and market inspection readiness, thus improving product management efficiency and sales strategies.

UserEntity and **UserDB**

- UserDB contains data that is needed to authorize system access and manage users together with their authentication data and profiles.
- UserEntity contains usernames together with encrypted passwords, roles and contact details, and preferences.
- The system enables secure and efficient user management, which supports authentication processes, authorization controls, and user experience customization features for the app.

2.2 Client

2.2.1 View

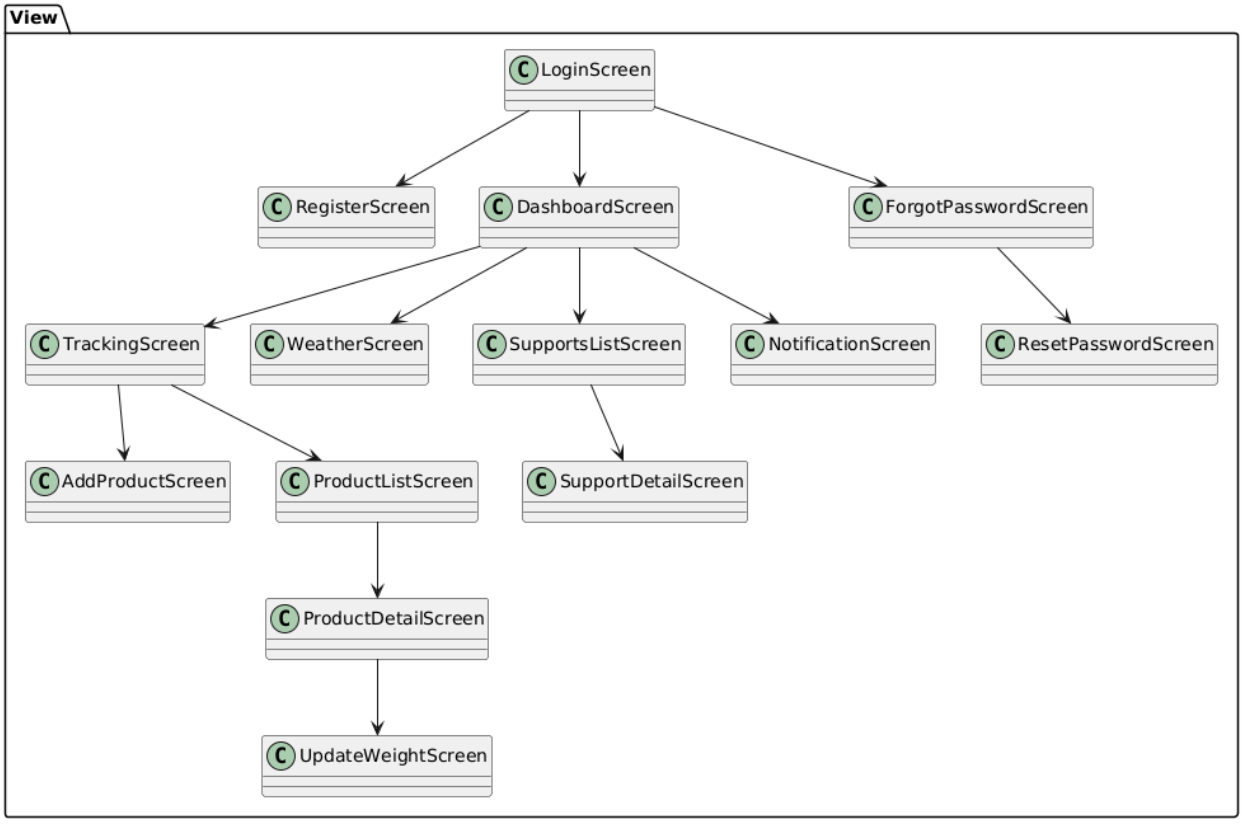


Figure 7: View Package

LoginScreen: Main entry for users to access to their accounts, features fields for username, password input, and mechanisms for authentication. Upon successful login, leads to DashboardScreen; if the user is in need of a password recovery, links to ForgotPasswordScreen; and if the user doesn't have an account, leads to RegisterScreen.

RegisterScreen: Its purpose is to allow users to create their accounts by providing necessary information (name, email, setting password). After a successful registration, it directs to LoginScreen.

DashboardScreen: Serves as a home screen of the app; users can access all functionalities. Directly sends users to TrackingScreen, WeatherScreen, SupportsListScreen, and NotificationScreen when they need to.

ForgotPasswordScreen and ResetPasswordScreen: The main purpose is to help users recover or reset their passwords. ForgotPasswordScreen provides a way to initiate the password recovery process, and ResetPasswordScreen allows users to set a new password after verifying their identity.

TrackingScreen: Displays tracking information for crops and livestock. Users can navigate to AddProductScreen through links from this screen to create new products from agricultural crops or livestock.

WeatherScreen: Shows current and forecasted weather conditions that are relevant to the user's direct location. Directly accessible from the DashboardScreen.

SupportsListScreen: Lists all available support programs. Accessed by DashboardScreen.

SupportDetailScreen: Shows complete information regarding a particular support program. Users can access this screen by choosing from the SupportsListScreen options.

NotificationScreen: This screen displays all notifications which pertain to the user's agricultural activities. Users can view notification details directly after accessing this screen.

ProductListScreen: Users can view product listings that align with their entered information on ProductListScreen. Users can access this screen by selecting a product from the ProductListScreen.

ProductDetailScreen: This screen displays complete details about a particular product.

AddProductScreen: Gives users the capability to create records for new products that stem from their agricultural work.

UpdateWeightScreen: Lets users modify existing product metrics which users can reach through ProductDetailScreen.

2.2.2 Service

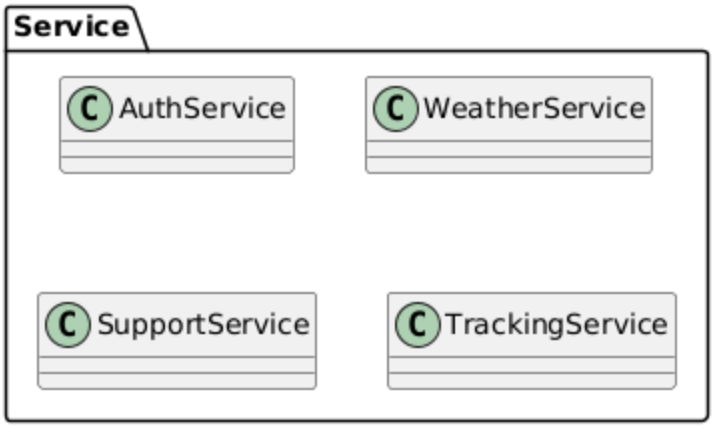


Figure 8: Client Services

AuthService

- The security functions related to user login, registration, and session management achieved through the use of authentication security with the AuthService are handled safely through the safe handling of the user data.
- In other words, authentication of a user verifies user credentials and gives security tokens such as JWT.
- The system handles user registration by verifying data and safely storing all registered information.
- The system verifies user authorization status for different sections within the program.

WeatherService

- Offers vital meteorological data that supports agricultural operations. The current and future weather information is obtained through external weather API communication from this service.
- The system implements a process that retrieves weather data from external sources and stores it in the cache.
- The system develops and distributes weather alerts directly to users who meet particular weather and location requirements.
- The system provides users with short-term and long-term weather forecasts that support their agricultural operational planning.

SupportService

- Handles interactions with government support programs. The system deals with user application management while tracking their workflow and notifying users about software updates.
- Support Program Listing maintains a comprehensive list of available support programs.
- The service processes all applications sent to support programs through its application processing system.
- The system provides users with live information about the current status of all their applications through support status updates.

TrackingService

- It monitors and documents various agricultural activities which include livestock management and crop growth. The service collects all data while generating analytical reports from recorded information.
- The system records agricultural data through its data logging capability.
- The system observes data, from an agricultural operation and with that creates recommendations and notifies on collected observations.
- Analytical reports are generated by the system whereby historical information and trends are combined to help users make better decisions.

2.2.3 Model

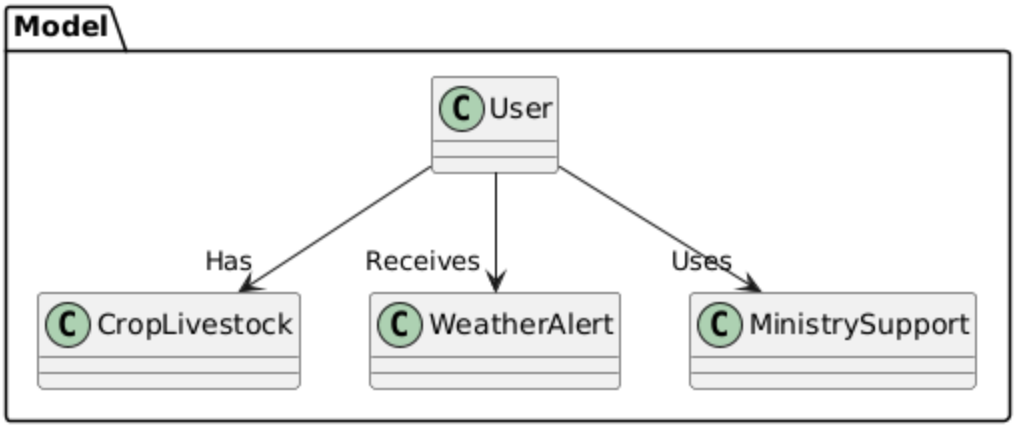


Figure 9: Client Model

- **User:** Represents the ALSApp's end consumers. This category includes farmers and livestock managers as well as administrative personnel working at the Ministry of Agriculture and Forestry. The CropLivestock system directly belongs to the User category through a “has-a” relationship. A User holds the role of ownership or crop and livestock management responsibilities. The entity WeatherAlert sends notifications and alerts to its associated users through the “receives” relationship. Through its “Uses” relationship with MinistrySupport, users gain the opportunity to benefit from government support programs
- **CropLivestock:** Allows to read information related to both crop and livestock by combining several models which deal with information of crop and livestock species types and breeds along with health parameters related to each. With this system, all the agricultural and livestock resource management is done by monitoring the health performance and feeding schedules, disease control and harvest planning operations.
- **WeatherAlert:** Contains weather-related information for agricultural operations. WeatherAlert also provides users with advance notice about dangerous weather conditions and enables them to take necessary precautions in a timely manner.
- **MinistrySupport:** Provides data about government support programs obtained from the Ministry of Agriculture and Forestry. The system provides financial support as well as product assistance to help users access multiple support programs, apply for them, and handle their benefits.

2.2.4 API

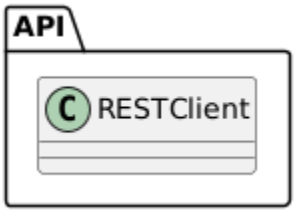


Figure 10: Client API

RESTClient is the API layer of ALSApp, also known as the fundamental component to handle all client-server communication processes. It manages server communication through HTTP requests, which include GET, POST, PUT, and DELETE operations. The data exchange system based on JSON format ensures secure communication while managing authentication tokens and providing continuous sessions alongside error handling capabilities. The app benefits from simplified direct server interaction management through this system because it centralizes server interactions and facilitates modular architecture while ensuring dependability and performance.

2.2.5 Widget

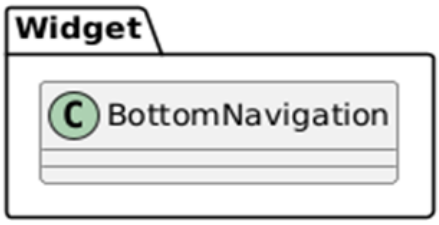


Figure 11: BottomNavigation Widget

BottomNavigation serves as a screen interface element that takes place at the bottom of mobile applications where users can navigate effortlessly between dashboard tracking, weather, support services, and other essential app components. Users experience seamless device compatibility through this interface because it features a simplified design scheme for all sections. User experience takes priority on the widget because its interface becomes more intuitive and users can easily navigate through the app features following modern design principles such as Google Material Design.

3. Class Interfaces

This section deals with the separation of the class interfaces into two parts which are then further explained. Here, the fact that this project has been made to increase the quality of the project for both the client and the server, the main aim is to describe all their responsibilities as much as possible. In these two parts, while the client side deals with user interactions and sends requests, the server side processes these requests and gives the appropriate responses or the data.

3.1. Server Side

In this section server-side class interfaces were demonstrated in detail, explaining how they process incoming information. They are covered in the following segments with examples, explaining how and with what methods and mechanisms they provide apt responses.

3.1.1 Models

Class CropLivestock
Essential information about a crop or a livestock entity like its name and type is provided by this class. It stands in for the difference in products between animals and plants in the application's agricultural management system. These details are encapsulated by the class to efficiently track, categorize, and handle many farm resources.
Attributes
• **String name:** The name of the crop or livestock.
• **String type:** The type or category included here (e.g., "Animal", "Crop").
Methods
• **Constructor:** CropLivestock({required this.name, required this.type})

Class MinistrySupport
This class is used to represent government or ministry backed support programs and its unique ID and description. This allows the system to be able to tell the user about subsidy and assistance programs available to them. This data can be used by users to explore and apply for the appropriate support opportunities.
Attributes
• **String id:** The unique identifier of the support program.
• **String description:** A basic description or details of the program.
Methods
• **Constructor:** MinistrySupport({required this.id, required this.description})

Class User
Here the essential attributes of this class that pertains to a user through his/her ID name and phone number are defined . It is a keystone component that acts as the base in which user management, authentication, and personalization in the app. The system can customize features and notifications to fit its users depending on the tracked details.
Attributes
• **String id:** Represents the user's unique identifier.
• **String name:** Represents the user's name.
• **String phone:** Represents the user's phone number (or other contact info).
Methods
• **Constructor:** User({required this.id, required this.name, required this.phone})

Class WeatherAlert
This class stores the location reached by the condition (e.g.,”Heavy Rain”) as an alert in this class. The alerts help to notify the users of potentially hazardous or important weather changes in specific regions. The application can warn immediately and provide appropriate advice to reduce the risks related to weather through this alert system.
Attributes
• **String cityName:** This attribute defines a default city name (e.g., “Konya”).
• **temperature, humidity, precipitation, riskLevel, weatherWarning :** The weather data retrieved via the API is stored here.
• **Boolean isLoading :** This attribute was utilized to show a loading spinner while data is being fetched.
• **LatLng mapLocation:** Coordinates for Google Maps display.
Methods
• **Constructor:** WeatherAlert({required this.condition, required this.location})

3.1.2 Services

Class AuthService
This class is responsible for managing user authentication by offering the login and registration functionality. It is able to be extended to add it or connect it to an actual backend or database, if necessary. Centralization of these operations provides a way to securely and consistently handle user credentials by the application.
Attributes
• **There are no unique properties; all actions are done through methods.**
Methods
• **Future<bool> login(String phone, String password):** This method was utilized to log in with the specified phone number and password.
• **Future<bool> register(String name, String phone, String password):** This method attempts to register a new user using the given information.

Class DummyWeatherService (from support_service.dart - a dummy version)
This is a dummy (placeholder) weather service class that always returns a fixed string of weather data. This shows how the application can query and show basic weather information. This can be replaced or extended to a real implementation for production use in the future implementations.
Attributes
• **No notable attributes, since it's a dummy service right now.**
Methods
• **Future<String> fetchWeather(String location):** This specific method returns an instance of the weather string (such as "Sunny, 25°C") that will be used for the specified location.

Class WeatherService (the real one in weather_service.dart)
Application retrieves the real weather data via an external API (AccuWeather, PlacesAPI etc.) using this class. It offers a method to obtain a location key for fetching a city's name, as well as for getting current weather conditions. The network requests and data parsing are handled by it thus allowing to integrate the live weather information into the app much easier.
Attributes
• **String apiKey :** The API key used to access the weather service.
• **String baseUrl :** The base URL for the API.
Methods
• **Future<String?> getLocationKey(String cityName):** Retrieves a location key for the specified city name from the API.
• **Future<Map<String, dynamic>?> getCurrentWeather(String locationKey):** This specific method gets current weather data (in JSON) for the specified location.

3.2 Client Side

This part provides an in-depth explanation of the client-side class interfaces, highlighting how they handle user requests. They are demonstrated below, showing various examples of their functionality and interaction with the system.

3.2.1 Screens

Class LoginScreen
This screen facilitates user authentication via phone or email and password inputs. It is the first point of entry into the app that redirects successful logins to the dashboard. An error occurs if the credentials are invalid, and only valid users are allowed to access.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context):** Builds the widget tree for the login form. After a successful login it will navigate to DashboardScreen.

Class RegisterScreen
Through this class, users could create an account with details: phone, password, etc. It guarantees to provide you valid and complete information to submit. After registering, the potential users can access the main features of the app.
Attributes
• **String occupation** : Holds the user's selected occupation (farmer or employee).
Methods
• **build(BuildContext context)**: Builds the screen UI and manages the registration form.

Class ForgotPasswordScreen
This screen allows users to require the password reset by inputting the email or phone number. So it sends a verification code to help initiate a secure recovery process. Having that done, users can go to the reset screen to set their new credentials.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context)**: Builds the UI and with "Send Code," navigate to ResetPasswordScreen.

Class ResetPasswordScreen
The second password recovery step is this screen. After that, users enter a verification code and select a different password. This is a way of ensuring the accounts stay safe from someone making changes to its accounts without authorization.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context)**: Builds the widget tree for the password reset form.

Class DashboardScreen
It is the main landing screen after login with rich information to aid you in accessing the notices and quick links. It serves as the gateway for the users to move on to different sections of the app. The dashboard consolidates important data to give a snapshot of the user's present condition.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context):** Builds the UI with notification cards (support notification, weather alert, tracking update, etc.).

Class NotificationScreen
After login it is the main landing screen where it displays data such as notifications and quick links. It is a gateway which lets the users reach other parts of the app. Depending on how the data is consolidated with the dashboard, the information it provides is a snapshot of the user's current situation.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context)**: Builds a simple interface showing a sample notification message.

Class ProductDetailScreen
The information shown in this screen is more detailed information of selected products or animals. It will feature description, stats, or recent activity etc. for a holistic view. Users can also edit or add new details.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context): Shows details and an “Add New Info” button.**

Class ProductListScreen
The formatting is outside of the list with a scrollable list of the users products or animals. Each item has a summary and a "More Info" button that allows the user to enter ProductDetailScreen. Organizing the items visually makes it easier for the user to manage their inventory.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context)**: Uses ListView.builder to list sample products/animals. "More Info" navigates to ProductDetailScreen.

Class AddProductScreen
The purpose of this screen is for users to add new items to this inventory, i.e. crops or livestock. Usually it contains form elements (dropdowns, text fields) for filling out the product details. The app constantly funnels the data from all the additions through a single data structure; therefore, it's consistent.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context)**: Builds the UI with a dropdown for "Animal" or "Product."

Class TrackingScreen
This screen shows sorts of product or livestock related functions into one narrow screen. It has short cuts for adding new items and viewing existing items. By putting all these operations into one place, this makes the user's work easier to keep track of farm or livestock assets.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context)**: Contains "Add Product or Animal" and "View Products/Animals" buttons.

Class UpdateWeightScreen
Through this screen, metrics such as an animal's weight or milk production can be updated. Typically, it has input fields for input of the new data and a save button. Such regular tracking allows farmers and their breeders to keep up to date, accurate records.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context)**: Contains two TextFields for weight and production, plus a save button.

Class SupportsListScreen
This screen presents an overview of the types of programs the Government has to offer support. They present each item in a card with all necessary information and link to more detailed information. Consolidating these programs makes the process of locating the relevant help simpler.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context)**: Uses ListView.builder to show sample supports. The "More Info" button navigates to SupportDetailScreen.

Class SupportDetailScreen
On this screen, we see detailed information about a specific program in which a government gives some support. There are also benefits, requirements and how to apply for users to review. It allows the clients to make decisions on which supports they would like to pursue.
Attributes
• **Integer supportIndex**: Indicates which support program (by index) the user is viewing.
Methods
• **build(BuildContext context)**: Displays details about the chosen support program.

Class WeatherScreen
Real-time weather information comprising temperature data as well as humidity values and precipitation reports is retrieved by the weather module for display. A map displaying user location or affected areas might be included by this screen. Actions can take place with the help of external weather services integrated into the application.
Attributes
• **String cityName :** Default city name (e.g., “Konya”).
• **Integer temperature, Integer humidity, Integer precipitation, Integer riskLevel, String weatherWarning :** Holds the API-fetched weather data.
• **Boolean isLoading :** Used to display a loading spinner while data is being fetched.
• **LatLng mapLocation :** Coordinates for Google Maps display.
Methods
• **initState()**:this function runs when the screen loads for the first time and retrieves weather data.
• **fetchWeatherData()**: this method invokes the WeatherService to retrieve API data that the screen requires.
• **build(BuildContext context)**: Builds the UI, displays the data, and includes a map widget.

3.3 Widgets and Main Application

Class BottomNavigation
The bottom navigation bar included in this widget is used to switch from app section to app section, such as Tracking, Supports, and Weather. It helps in keeping the user interface manageable, simple and easy to navigate. The widget routes a user to the associated screen when a user taps an icon.
Attributes
• **Integer currentIndex** : Determines which tab is currently selected.
Methods
• **_onItemTapped(int index):** Updates the selected tab and navigates to the corresponding screen.
• **build(BuildContext context):** Builds the bottom navigation bar.

Class ALSApp (main.dart)
This is the application's startup root widget. This seasons the primary theme, routes, and the initial screen (LoginScreen). Moreover, Gathering configuration here controls the user experience across all screens.
Attributes
• **(it is a StatelessWidget so it has no specified class-level fields.)**
Methods
• **build(BuildContext context):** Provides the main configuration of the app and displays the login screen.

4. References

[1] "What is Low-Level Design Document?" GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/what-is-low-level-design-document/. [Accessed: Mar. 9, 2025].

[2] "Package Diagram: Introduction, Elements, Use Cases, and Benefits," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/package-diagram-introduction-elements-use-cases-and-benefits/. [Accessed: Mar. 9, 2025].

[3] "What is Package Diagram?" Visual Paradigm. [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/. [Accessed: Mar. 9, 2025].

[4] E. S. Dede, S. D. Demirci, and G. Ünsal, "ALSApp: Agriculture and Livestock Support Application - Analysis Report," TED University, Department of Computer Engineering, 2024.

[5] E. S. Dede, S. D. Demirci, and G. Ünsal, "ALSApp: Agriculture and Livestock Support Application - High Level Design Report," TED University, Department of Computer Engineering, 2024.

[6] Association for Computing Machinery (ACM), "ACM Code of Ethics and Professional Conduct," 2018. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: 28-Feb-2025].

[7] Bilkent MuSync Team, "Musync - Low-Level Design Report," Bilkent University, 2024. [Online]Available:https://bilkent-musync.github.io/docs/Musync%20-%20Low-Level%20Design%20Report.pdf.