AgroLink: Enhancing Crop Management

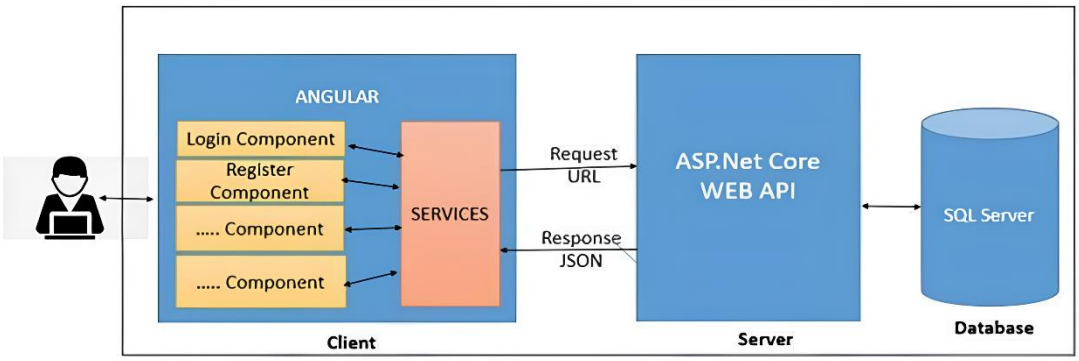
Introduction:

The AgroLink: Enhancing Crop is a user-friendly web application designed to streamline interactions between farmers and sellers in the agricultural sector. With the AgroLink, farmers can efficiently request agrochemicals for their crops, manage crop details, and track the status of their requests. Sellers, on the other hand, can easily manage their inventory of agrochemicals, respond to farmers' requests. By providing a centralized platform for crop-related activities, the AgroLink aims to enhance collaboration, communication, and productivity within the agricultural community.

Users of the System:

1. Farmer
2. Seller

System Architectural Diagram:



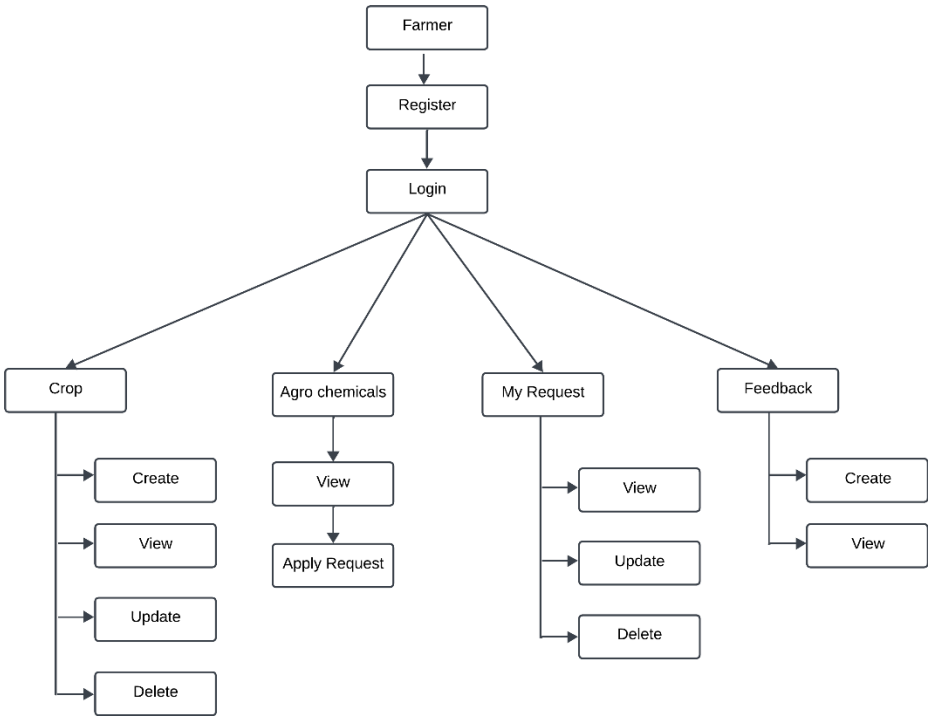
Application Flow:

User Authentication and Authorization:

- Users should be able to register an account with a unique email and password.
- Users should be able to log in securely using their credentials.
- Different user roles should have different levels of access (farmer, seller).

Farmer Actions:

Flow Diagram:



- **Crop Management:**
  - Create Crop: Farmers can add new crop entries to the system, providing details such as crop name, type, description, and planting date.
  - Read Crop: View the details of existing crops, including their name, type, description, and planting date.
  - Update Crop: Modify the details of existing crops, such as updating the planting date or description.
  - Delete Crop: Remove crop entries from the system that are no longer relevant or needed.

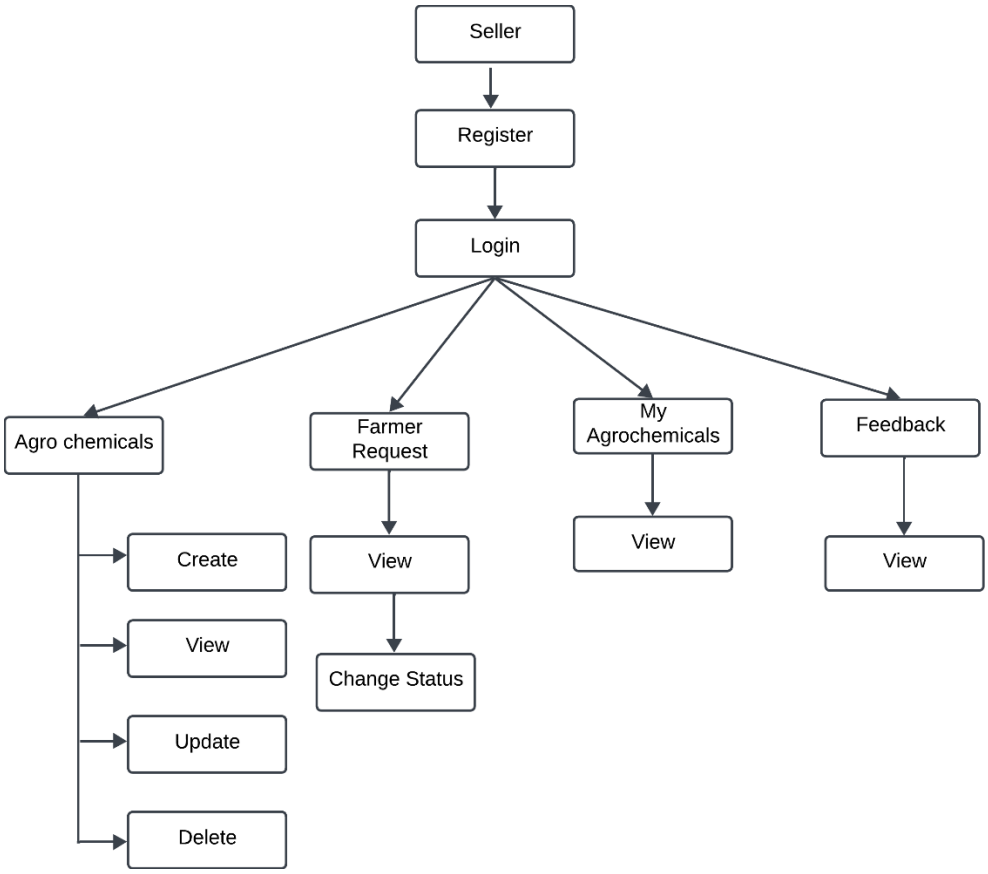
- **Agrochemical Management:**
  - View Agrochemicals: Access a list of available agrochemicals along with details like name, brand, category, quantity, and price per unit.

- **Request Management:**
  - Apply for Agrochemical Request: Submit requests for required agrochemicals, specifying the quantity needed and the purpose of the request.
  - View Request Status: Check the status of submitted agrochemical requests, whether they are pending, accepted, or rejected.
  - Update Request: Modify details of agrochemical requests, such as changing the quantity or purpose.
  - Delete Request: Cancel agrochemical requests that are no longer necessary or relevant.

- **MyRequests:**
    - View a list of all requests submitted by the farmer, including details such as request ID, crop ID, requested agrochemical, quantity, request purpose, and status.
- **Feedback:**
    - Farmer should have the option to view and post their feedback.

Seller Actions:

Flow Diagram:



- **Agrochemical Management:**
    - Add Agrochemical: Sellers can add new agrochemicals to their inventory, providing details such as name, brand, category, quantity, price per unit, and suitable crop.
    - View Agrochemicals: Access a list of available agrochemicals in their inventory, including details like name, brand, category, quantity, and price per unit.
    - Update Agrochemical: Modify details of existing agrochemical entries, such as updating the quantity or price per unit.
    - Delete Agrochemical: Remove agrochemical entries from their inventory that are no longer available or stocked.

- **Request Management:**
    - View Incoming Requests: Access a list of requests submitted by farmers for agrochemicals, including details like crop ID, requested agrochemical, quantity, and request status.
    - Change Request Status: Update the status of incoming requests, marking them as accepted, rejected, or pending based on availability and fulfilment criteria.
- **Feedback:**
    - Sellers should have the option to view all the farmer's feedback.

Data Security with JWT (JSON Web Tokens):

- **Token Generation and Validation:** Generate JWT tokens securely upon successful user authentication. Validate JWT tokens on each request to ensure their authenticity.
- **Authentication and Authorization:** Implement JWT-based authentication and ensure that only authenticated users with valid JWT tokens can access sensitive data and perform authorized actions.

Error and Exception handling:

Employ try and catch blocks to manage potential risks, particularly during an Axios call. If an error is detected, the objective is to redirect to the page that displays the error message.

Models:

User model:

1. **userId**: This field represents the primary key for each record in the user table. It uniquely identifies each record. This field is required.
2. **username**: A unique string representing the username of the user. It is required and must be unique. This field is required.
3. **password**: A string representing the password of the user. It is required for user authentication. This field is required.
4. **role**: A string representing the role of the user (e.g., employee, manager). It is required to determine user permissions. This field is required.
5. **email**: A string representing the email address of the user. It is required and must be unique. This field is required.
6. **mobileNumber**: A string representing the mobile number of the user. It is required for contact purposes. This field is required.
7. **profileImage:** A string representing the profile image of the user encoded in base64 format. It can be used to store the image directly as a string. This field is required.

Crop model:

1. **cropId** (Primary Key): Unique identifier for the crop. This field uniquely identifies each crop record in the table. It is required.
2. **cropName**: A string representing the name of the crop. This field is required.

3. **cropType**: A string representing the type or category of the crop (e.g., vegetables, fruits, grains). This field is required.
4. **description**: A string representing the description of the crop. This field provides additional details about the crop. It is required.
5. **plantingDate**: The date when the crop was planted. This field stores the planting date of the crop. It is required and of type Date.

AgroChemicals model:

1. **agrochemicalId** (Primary Key): Unique identifier for the agrochemical. This field uniquely identifies each agrochemical record in the table. It is required.
2. **name**: A string representing the name of the agrochemical. This field is required.
3. **brand**: A string representing the brand of the agrochemical. This field provides information about the manufacturer or brand of the product. It is required.
4. **category**: A string representing the category or type of the agrochemical (e.g., pesticide, herbicide, fertilizer). This field categorizes the agrochemical. It is required.
5. **suitableCrop**: A string representing the name of the crop for which the agrochemical is suitable. This field specifies the crop type that the agrochemical is intended for. It is required.
6. **description**: A string representing the description of the agrochemical. This field provides additional details about the agrochemical, such as its composition or usage instructions. It is required.
7. **quantity**: An integer representing the quantity of the agrochemical available. This field indicates the available quantity of the agrochemical in stock. It is required.
8. **pricePerUnit**: A floating-point number representing the price per unit of the agrochemical. This field indicates the cost of a single unit of the agrochemical. It is required.
9. **image**: A string representing the URL to the image of the agrochemical. This field stores the image URL of the agrochemical for visual reference. It is required.

Request model:

1. **requestId** (Primary Key): Unique identifier for the request. This field uniquely identifies each request record in the table. It is required.
2. **userId** (Foreign Key referencing User): Identifier of the user making the request. This field references the userId from the User Model. It is required.
3. **cropId** (Foreign Key referencing Crop): Identifier of the crop for which the request is made. This field references the cropId from the Crop Model. It is required.
4. **agrochemicalId** (Foreign Key referencing AgroChemicals): Identifier of the agrochemical requested. This field references the chemicalId from the AgroChemicals Model. It is required.
5. **quantity**: Quantity of the agrochemical requested. This field specifies the quantity of the agrochemical requested by the user. It is required.
6. **requestPurpose**: Reason for the request. This field provides a description or purpose for the requested agrochemical. It is optional.

7. **status**: Status of the request (e.g., pending, accepted, rejected). This field indicates the current status of the request. It is required.
8. **requestDate**: Date when the request was made. This field stores the date and time when the request was created. It is required.

Feedback model:

1. **feedbackId:** This field represents the primary key for each record in the feedback table. It uniquely identifies each record. This field is required.
2. **userId:** The unique identifier of the user associated with the feedback. It references the User model and is required to associate the request with a specific user. This field is required.
3. **feedback:** Feedback provided by the user. This field is required.
4. **date**: The date for which the feedback is recorded. It is of type Date and is required.

Controllers:

These controllers manage different aspects of user, crop, agrochemicals, and request within the system.

1. User Controller
2. Crop Controller
3. Agrochemicals Controller
4. Request Controller
5. Feedback Controller

Endpoints:

1. Backend URL + **"/user/login**":
    a. Description: Log in a user with provided credentials.
    b. Method name: POST
    c. Request Body:
    Contains the following fields:
        i. Username: The username of the user.
        ii. Password: The password associated with the username.
    d. Response:
        i. Success (200 OK): Contains a JWT token and user details if the login is successful.
        ii. Error (401 Unauthorized): Returns an error message if the provided credentials are invalid.

2. Backend URL + **"/user/register"**:
    a. Description: Register a new user with provided details.
    b. Method name: POST
    c. Request Body:
    Contains the following fields:
        i. Username: The desired username for the new user.

ii. Password: The password for the new user account.
iii. Email: The email address of the new user.
iv. Role: The role or type of the new user account (e.g., employee, manager).
v. Mobile Number: The mobile number of the new user.
vi. Profile Image: The profile image of the new user.

d. Response:
i. Success (200 Ok): Indicates that the user has been successfully registered along with user details.
ii. Error (400 Bad Request): Returns an error message describing the reason for failure in case of registration error.

3. Backend URL + **"/user/getAllFarmers"**:
a. Description: Retrieve all farmers.
b. Method name: POST
c. Request Body:
Contains the following fields:
i. SearchValue
ii. SortValue
iii. PaginationValue
d. Response:
i. Success (200 Ok): List of all farmers.
ii. Error (400 Bad Request): Returns an error message describing the reason for failure.

4. POST BackendURL + **"/crop/addCrop"**
a. Description: Add a new crop.
b. Method: POST
c. Request Body: Contains the details of the new crop.
d. Response:
i. Success (200 Ok): Details of the newly added crop.
ii. Error (400 Bad Request): If the request body is invalid.

5. GET BackendURL + **"/crop/getCropByCropID/:cropId"**
a. Description: Get a specific crop by its ID.
b. Method: GET
c. Parameter: cropId
d. Request Body: None
e. Response:
i. Success (200 Ok): Details of the specific crop.
ii. Error (404 Not Found): If the crop with the specified ID is not found.

6. PUT BackendURL + **"/crop/updateCropByCropID/:cropId"**
a. Description: Update an existing crop by its ID.
b. Method: PUT

c. Parameter: cropId
d. Request Body: Contains the updated details of the crop.
e. Response:
    i. Success (200 Ok): Details of the updated crop.
    ii. Error (400 Bad Request): If the request body is invalid.

7. DELETE BackendURL + **"/crop/deleteCropByCropID/:cropId"**
    a. Description: Delete a crop by its ID.
    b. Method: DELETE
    c. Parameter: cropId
    d. Request Body: None
    e. Response:
        i. Success (200 Ok): Message confirming successful deletion of the crop.
        ii. Error (404 Not Found): If the crop with the specified ID is not found.

8. GET BackendURL + **"/crop/getAllCrop"**
    a. Description: Retrieve all crops.
    b. Method: GET
    c. Request Body:
        i. Search: (Optional) Search query for filtering crops.
        ii. Sort: (Optional) Sorting criteria for the retrieved crops.
        iii. Pagination: (Optional) Pagination parameters for fetching a subset of crops.
    d. Response:
        i. Success (200 Ok): List of all crops.
        ii. Error (404 Not Found): If no crops are found.

9. GET BackendURL + **"/agrochemical/getAgrochemicalsByUserId/:userId"**
    a. Description: Retrieve agrochemicals by user ID.
    b. Method: GET
    c. Parameter: userId
    d. Request Body: None
    e. Response:
        i. Success (200 Ok): List of agrochemicals for the specified user ID.
        ii. Error (404 Not Found): If no agrochemicals are found for the user ID.

10. POST BackendURL + **"/agrochemical/addAgrochemical"**
    a. Description: Add a new agrochemical.
    b. Method: POST
    c. Request Body: Contains the details of the new agrochemical.
    d. Response:
        i. Success (200 Ok): Details of the newly added agrochemical.
        ii. Error (400 Bad Request): If the request body is invalid.

11. GET BackendURL + **"/agrochemical/getAgrochemicalByAgrochemicalID/:agrochemicalId"**

a. Description: Get a specific agrochemical by its ID.
b. Method: GET
c. Parameter: agrochemicalId
d. Request Body: None
e. Response:
    i. Success (200 Ok): Details of the specific agrochemical.
    ii. Error (404 Not Found): If the agrochemical with the specified ID is not found.

12. PUT BackendURL + **"/agrochemical/updateAgrochemicalByAgrochemicalID/:agrochemicalId"**
    a. Description: Update an existing agrochemical by its ID.
    b. Method: PUT
    c. Parameter: agrochemicalId
    d. Request Body: Contains the updated details of the agrochemical.
    e. Response:
        i. Success (200 Ok): Details of the updated agrochemical.
        ii. Error (400 Bad Request): If the request body is invalid.

13. DELETE BackendURL + **"/agrochemical/deleteAgrochemicalByAgrochemicalID/:agrochemicalId"**
    a. Description: Delete an agrochemical by its ID.
    b. Method: DELETE
    c. Parameter: agrochemicalId
    d. Request Body: None
    e. Response:
        i. Success (200 Ok): Message confirming successful deletion of the agrochemical.
        ii. Error (404 Not Found): If the agrochemical with the specified ID is not found.

14. GET BackendURL + **"/agrochemical/getAllAgrochemicals"**
    a. Description: Retrieve all agrochemicals.
    b. Method: GET
    c. Request Body:
        i. Search: (Optional) Search query for filtering agrochemicals.
        ii. Sort: (Optional) Sorting criteria for the retrieved agrochemicals.
        iii. Pagination: (Optional) Pagination parameters for fetching a subset of agrochemicals.
    d. Response:
        i. Success (200 Ok): List of all agrochemicals.
        ii. Error (404 Not Found): If no agrochemicals are found.

15. GET BackendURL + **"/request/getRequestByUserId/:userId"**
    a. Description: Retrieve requests by user ID.
    b. Method: GET

c. Parameter: userId
d. Request Body: None
e. Response:
    i. Success (200 Ok): List of requests for the specified user ID.
    ii. Error (404 Not Found): If no requests are found for the user ID.

16. GET BackendURL + **"/request/getAllRequest"**
    a. Description: Retrieve requests by user ID.
    b. Method: GET
    c. Request Body: None
    d. Response:
        i. Success (200 Ok): List of requests for the specified user ID.
        ii. Error (404 Not Found): If no requests are found for the user ID.

17. POST BackendURL + **"/request/addRequest"**
    a. Description: Add a new request.
    b. Method: POST
    c. Request Body: Contains the details of the new request.
    d. Response:
        i. Success (200 Ok): Details of the newly added request.
        ii. Error (400 Bad Request): If the request body is invalid.

18. GET BackendURL + **"/request/getRequestByRequestId/:requestId"**
    a. Description: Get a specific request by its ID.
    b. Method: GET
    c. Parameter: requestId
    d. Request Body: None
    e. Response:
        i. Success (200 Ok): Details of the specific request.
        ii. Error (404 Not Found): If the request with the specified ID is not found.

19. PUT BackendURL + **"/request/updateRequestByRequestId/:requestId"**
    a. Description: Update an existing request by its ID.
    b. Method: PUT
    c. Parameter: requestId
    d. Request Body: Contains the updated details of the request.
    e. Response:
        i. Success (200 Ok): Details of the updated request.
        ii. Error (400 Bad Request): If the request body is invalid.

20. DELETE BackendURL + **"/request/deleteRequestByRequestId/:requestId"**
    a. Description: Delete a request by its ID.
    b. Method: DELETE
    c. Parameter: requestId
    d. Request Body: None
    e. Response:

i. Success (200 Ok): Message confirming successful deletion of the request.
ii. Error (404 Not Found): If the request with the specified ID is not found.

21. Backend URL + **"/feedback/addFeedback":**
    a. Description: Add a new feedback.
    b. Method name: POST
    c. Request Body:
        i. Contains the details of the new feedback.
    d. Response:
        i. Success (200 Ok): Details of the newly added feedback.
        ii. Error (400 Bad Request): Returns an error message describing the reason for failure.

22. Backend URL + **"/feedback /getFeedbacksByUserId/:userId":**
    a. Description: Retrieve all the feedback specific to the user.
    b. Method name: POST
    c. Parameter: userId
    d. Request Body:
        i. No request body required.
    e. Response:
        i. Success (200 Ok): List of feedbacks for the specified user ID.
        ii. Error (400 Bad Request): Returns an error message describing the reason for failure.

23. Backend URL + **"/feedback /getAllFeedbacks":**
    a. Description: Retrieve all feedbacks.
    b. Method name: GET
    c. Request Body:
        i. No request body required.
    d. Response:
        i. Success (200 Ok): List of all feedbacks.
        ii. Error (400 Bad Request): Returns an error message describing the reason for failure.

Conclusion:

AgroLink: Enhancing Crop is a user-friendly web application designed to facilitate seamless interaction between farmers and sellers in the agricultural sector. By enabling farmers to efficiently manage crop details, request agrochemicals, and track request statuses, and empowering sellers to manage their agrochemical inventory and respond to farmers' requests, AgroLink aims to enhance collaboration, communication, and productivity within the agricultural community. Overall, AgroLink serves as a centralized platform for crop-related activities, promoting efficiency, transparency, and cooperation in the agricultural sector.