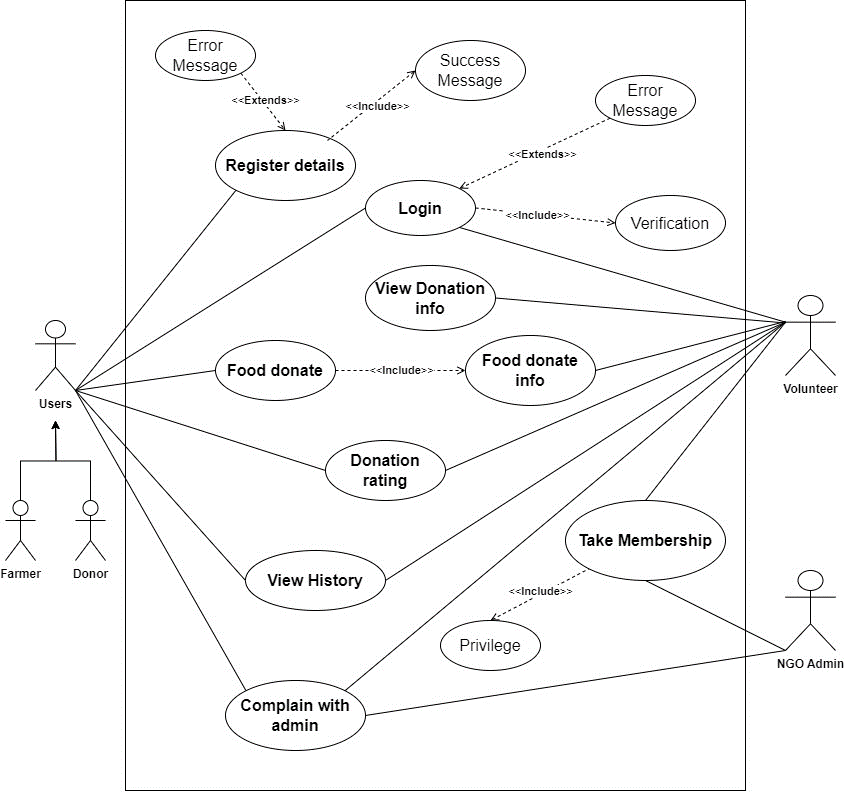
**Use Case Diagram**

|  |  |
| --- | --- |
| **Symbol** | **Name** |
| 🛉 | Actor |
|  | Use Case |
| ⸏ | Connector |
| ⟶ | Generalization |
| ⇢ | Stereotype |



**High-Level Use Case**

**Register**

|  |  |
| --- | --- |
| **Use Case:** | Register Details |
| **Actors:** | Donor, Volunteer, NGO, Farmer |
| **Descriptions:** | All users can input their respective details into the system. Upon submission, the system automatically registers the provided information, ensuring seamless integration and efficient data management. |

|  |  |
| --- | --- |
| **Use Case:** | Success Message |
| **Actors:** | Donor, Volunteer, Farmer |
| **Descriptions:** | The user can provide the details and go to register if it is valid then show the successful message. |

|  |  |
| --- | --- |
| **Use Case:** | Error Message |
| **Actors:** | Donor, Volunteer, Farmer |
| **Descriptions:** | The user can provide the details and go to register if it is not valid then show the error message. |

**Take Membership**

|  |  |
| --- | --- |
| **Use Case:** | Take Membership |
| **Actors:** | Volunteer, NGO |
| **Descriptions:** | A new volunteer provides the personal details, and his/her details are registered with the system. The NGO provide the membership, and then volunteers take the new membership. |

|  |  |
| --- | --- |
| **Use Case:** | Privilege |
| **Actors:** | Volunteer |
| **Descriptions:** | After taking a new member, a new volunteer gets the privilege of the food donation system. |

**Login**

|  |  |
| --- | --- |
| **Use Case:** | Login |
| **Actors:** | Donor, Volunteer, Farmer |
| **Descriptions:** | After registering details in the system, all the users can provide valid details and log in to the system Then successfully log in to the system. |

|  |  |
| --- | --- |
| **Use Case:** | Verification |
| **Actors:** | Donor, Volunteer, Farmer |
| **Descriptions:** | A new user-provided login details are checked in the database where it is registered or not registered in the system. If its details are registered, then verify to give the login access in the system. |

|  |  |
| --- | --- |
| **Use Case:** | Error Message |
| **Actors:** | Donor, Volunteer, Farmer |
| **Descriptions:** | If the user login details are not registered or invalid, then the system shows the error message. |

**Food Donate**

|  |  |
| --- | --- |
| **Use Case:** | Food Donate |
| **Actors:** | Donor, Farmer |
| **Descriptions:** | The Donor or Farmer can donate the proper food information and details with location. The system can show the donation food details in the history after posting the donated food. |

|  |  |
| --- | --- |
| **Use Case:** | Food Donate Information |
| **Actors:** | Donor, Farmer, Volunteer |
| **Descriptions:** | When a donor donates food posted in the system all the volunteers can get the donation information (Notification). |

**View Donation info**

|  |  |
| --- | --- |
| **Use Case:** | Donation Rating |
| **Actors:** | Volunteer |
| **Descriptions:** | After receiving the donation info, the volunteer can view the donation details if it is possible or not possible to distribute. |

**Donation Rating**

|  |  |
| --- | --- |
| **Use Case:** | Donation Rating |
| **Actors:** | Volunteer, Donor |
| **Descriptions:** | After the food is completely donated to some people the volunteer can give the donation rating to the donor with food distributed information. |

**View History**

|  |  |
| --- | --- |
| **Use Case:** | View History |
| **Actors:** | Donor, Volunteer, NGO, Farmer |
| **Descriptions:** | All the users can view the history of food donation where who donate, where to donate or more details. |

**Complain with Administration**

|  |  |
| --- | --- |
| **Use Case:** | Complain with Admin |
| **Actors:** | Donor, Volunteer, Admin |
| **Descriptions:** | After contact with donors and volunteers the donors cannot be provided or donate food and the receiver cannot come to receive the donation food, they can complain to the admin |

**Expanded Use Case**

|  |  |  |
| --- | --- | --- |
| **Use Case:** | Register Details | |
| **Actors:** | Donor, Volunteer, NGO, Farmer | |
| **Descriptions:** | All users can input their respective details into the system. The system automatically registers the provided information upon submission, ensuring seamless integration and efficient data management. | |
| Typical Courses of Events: | | |
| **Donor, Volunteer, NGO, Farmer** | | **System Response** |
| 1. A new user can provide the personal details for the register in the system. | |  |
|  | | 2. The system checks whether the provided details are valid or not. |
| 3. Request for register in the system. | |  |
|  | | 4. Conform register with a success message if no valid data, then show the error message. |

|  |  |  |
| --- | --- | --- |
| **Use Case:** | Take Membership | |
| **Actors:** | Volunteer, NGO | |
| **Descriptions:** | A new volunteer provides the personal details, and his/her details are registered with the system. The NGO provide the membership, and then volunteers take the new membership. | |
| Typical Courses of Events: | | |
| **Volunteer, NGO** | | **System Response** |
| 1. A new volunteer can provide the personal details for the registered membership in the NGO. | |  |
|  | | 2. Check the Volunteer details. |
| 3. Request to take membership in the NGO. | |  |
|  | | 4. Confirm the registered membership and give the privilege. |

|  |  |  |
| --- | --- | --- |
| **Use Case:** | Login | |
| **Actors:** | Donor, Volunteer, Farmer | |
| **Descriptions:** | After registering details in the system, all the users can provide valid details and log in to the system Then successfully log in to the system. | |
| Typical Courses of Events: | | |
| **Donor, Volunteer, Farmer** | | **System Response** |
| 1. A new user can provide the login details in the system. | |  |
|  | | 2. The system checks whether the provided details are valid or not. |
| 3. Request for login in the system. | |  |
|  | | 4. The system can navigate to the dashboard if valid data is provided to log otherwise shows the error message. |

|  |  |  |
| --- | --- | --- |
| **Use Case:** | Food Donation | |
| **Actors:** | Donor, Farmer, volunteers | |
| **Descriptions:** | The Donor or Farmer can donate the proper food information and details with location. The system can show the donation food details in the history after posting the donated food. | |
| Typical Courses of Events: | | |
| **Donor, Farmer** | | **Volunteers** |
| 1. A donor can post the food details for donation. | |  |
|  | | 2. After getting the donation information volunteers respond to the request to accept the food. |
| 3. Donors confirm to provide the food when contacted with the donor. | |  |
|  | | 4. Volunteers receive the food and go to distribute it. |

|  |  |  |
| --- | --- | --- |
| **Use Case:** | View Donation Info | |
| **Actors:** | Volunteers | |
| **Descriptions:** | All users can input their respective details into the system. The system automatically registers the provided information upon submission, ensuring seamless integration and efficient data management. | |
| Typical Courses of Events: | | |
| **Volunteer** | | **System Response** |
| 1. A volunteer can view the donation post food details. | |  |
|  | | 2. show the donation food details with all information like food info, donor info and location details. |
| 3. View the donated location. | |  |
|  | | 4. The system can show the food donated location with Google Maps. |

|  |  |  |
| --- | --- | --- |
| **Use Case:** | Donation Rating | |
| **Actors:** | Donor, Volunteer, Farmer | |
| **Descriptions:** | After the food is completely donated to some people the volunteer can give the donation rating to the donor with food distributed information. | |
| Typical Courses of Events: | | |
| **Volunteer, Farmer** | | **Donor** |
| 1. After the food received to distributed give the donation rating with all the information. | |  |
|  | | 2. Get the donated rating info. |
| 3. Not possible to accept the donation of food if the volunteer cannot give a response. | |  |
|  | | 4. Volunteers do not respond to receive the food and when the expiration date is over the donor gets the expiration notification. |

|  |  |  |
| --- | --- | --- |
| **Use Case:** | Complain with Admin | |
| **Actors:** | Donor, Volunteer, Farmer | |
| **Descriptions:** | All users can input their respective details into the system. The system automatically registers the provided information upon submission, ensuring seamless integration and efficient data management. | |
| Typical Courses of Events: | | |
| **Donor, Volunteer, Farmer** | | **Admin Response** |
| 1. The food donation time donor volunteer and farmer can complain to the admin. | |  |
|  | | 2. Admin verifies the complaint. |
| 3. Request for succession. | |  |
|  | | 4. Admin gives the warning. |
| 5. Received warning. | |  |

**Context Diagram**

A diagram of food donation system

Description automatically generated

Figure : Context diagram

**Data Flow Diagram (DFD)**

**Level-1**

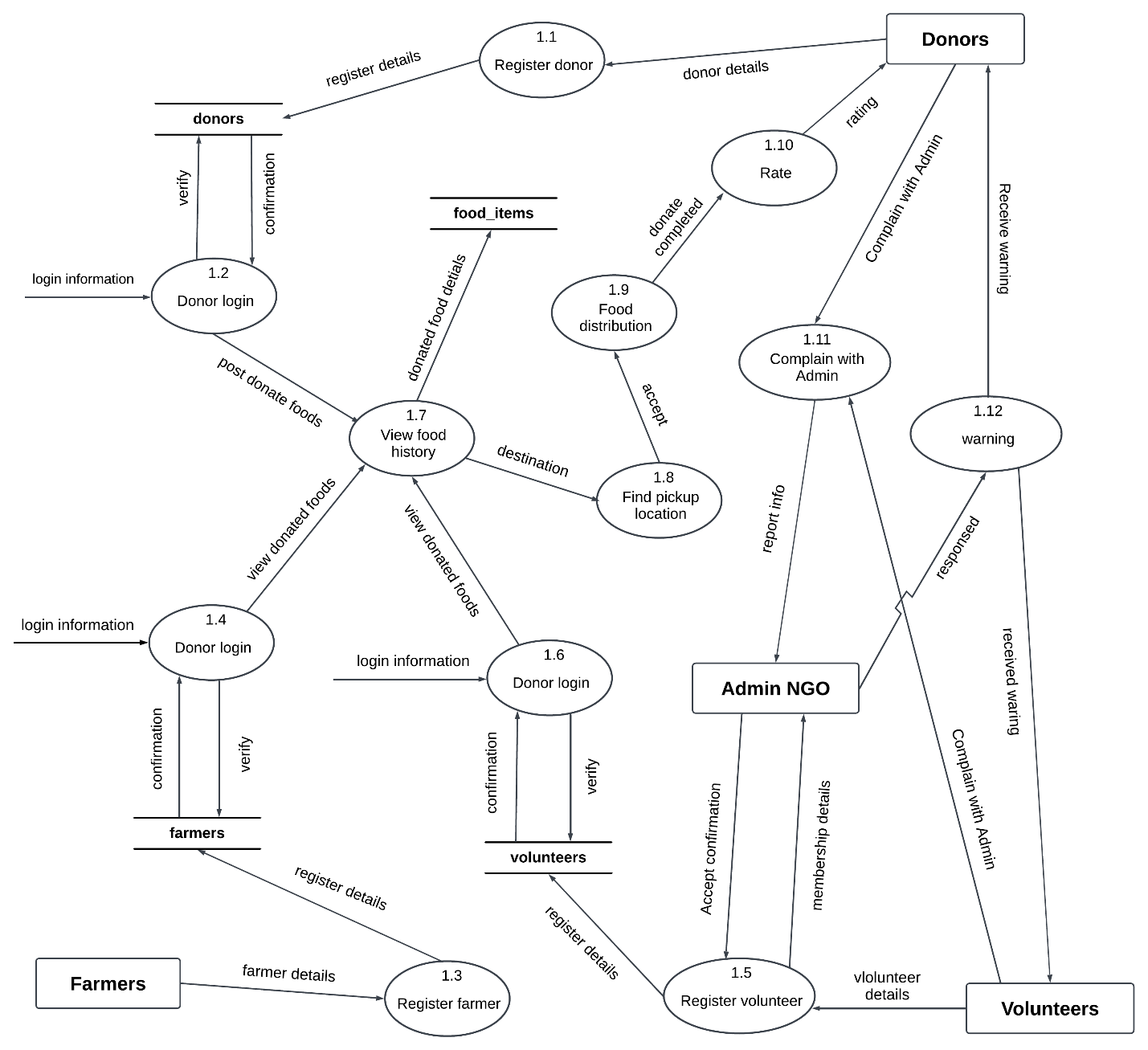
****

Figure 2: DFD Level-1

**Level-2**

**Register Details**

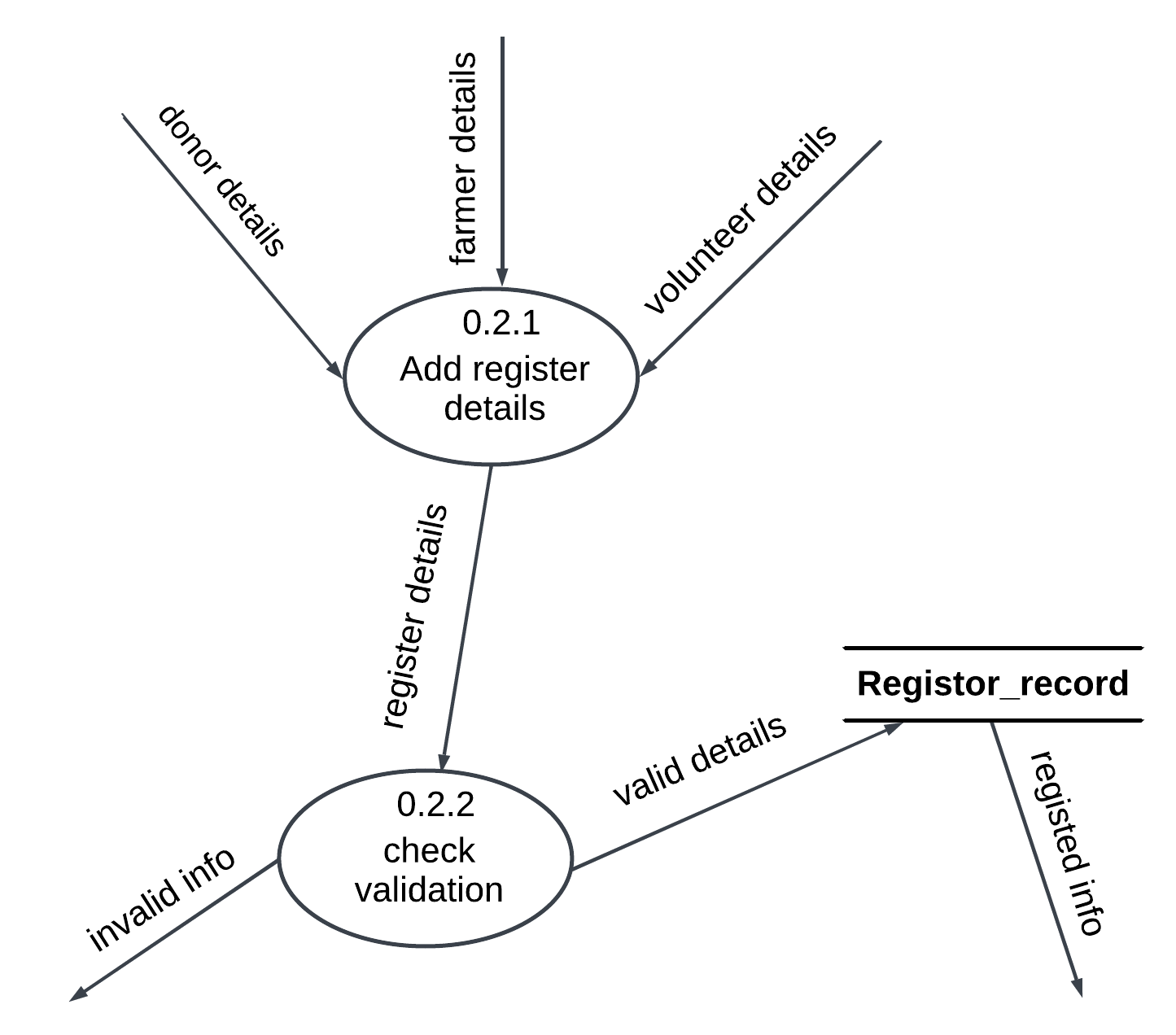


Figure 3: Register details DFD Level-2

**Take Membership**

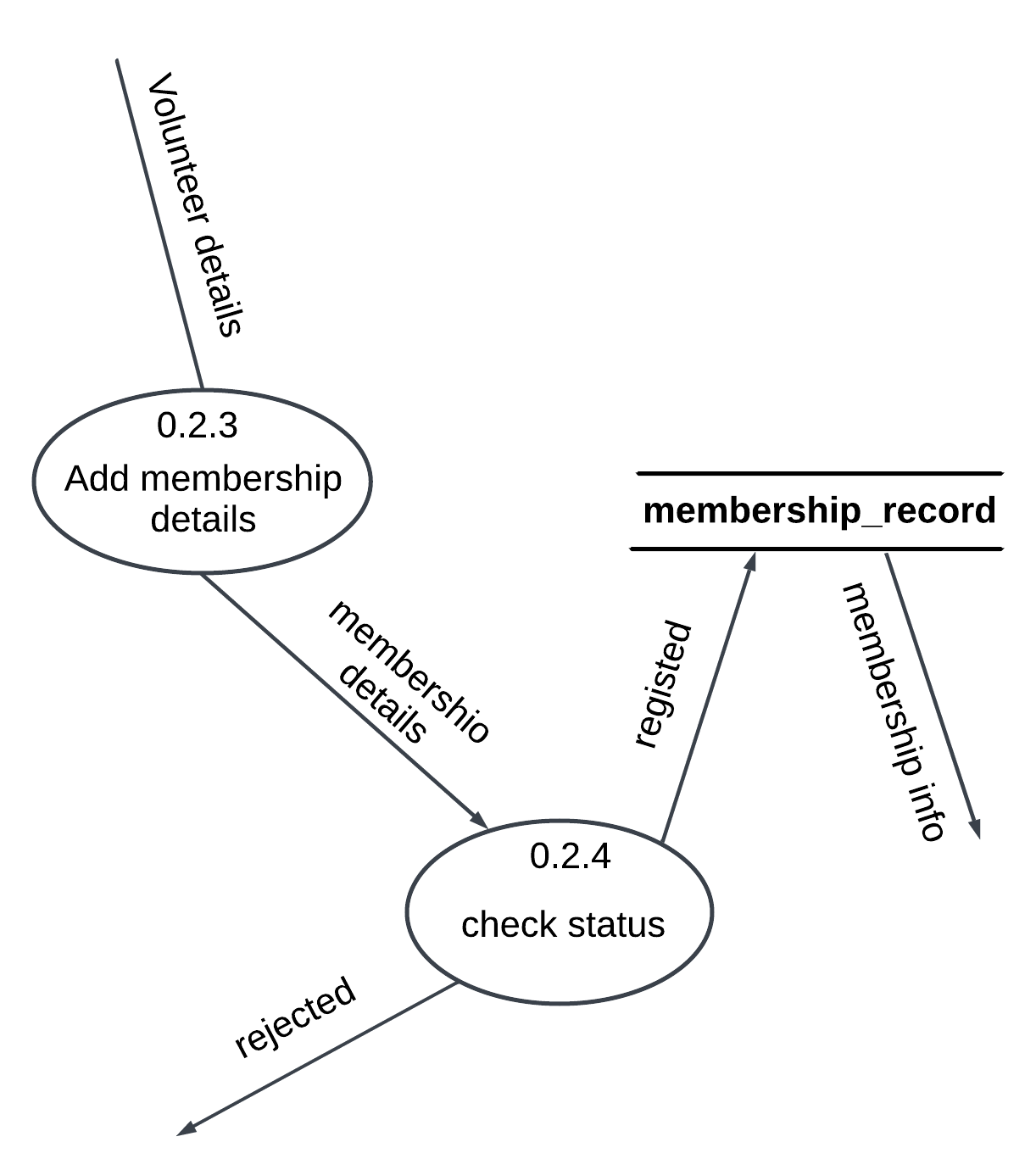


Figure 4: Take membership details DFD Level-2

**Login System**

A diagram of a member login system

Description automatically generated

Figure 5: Login details DFD Level-2

**Food Donate**

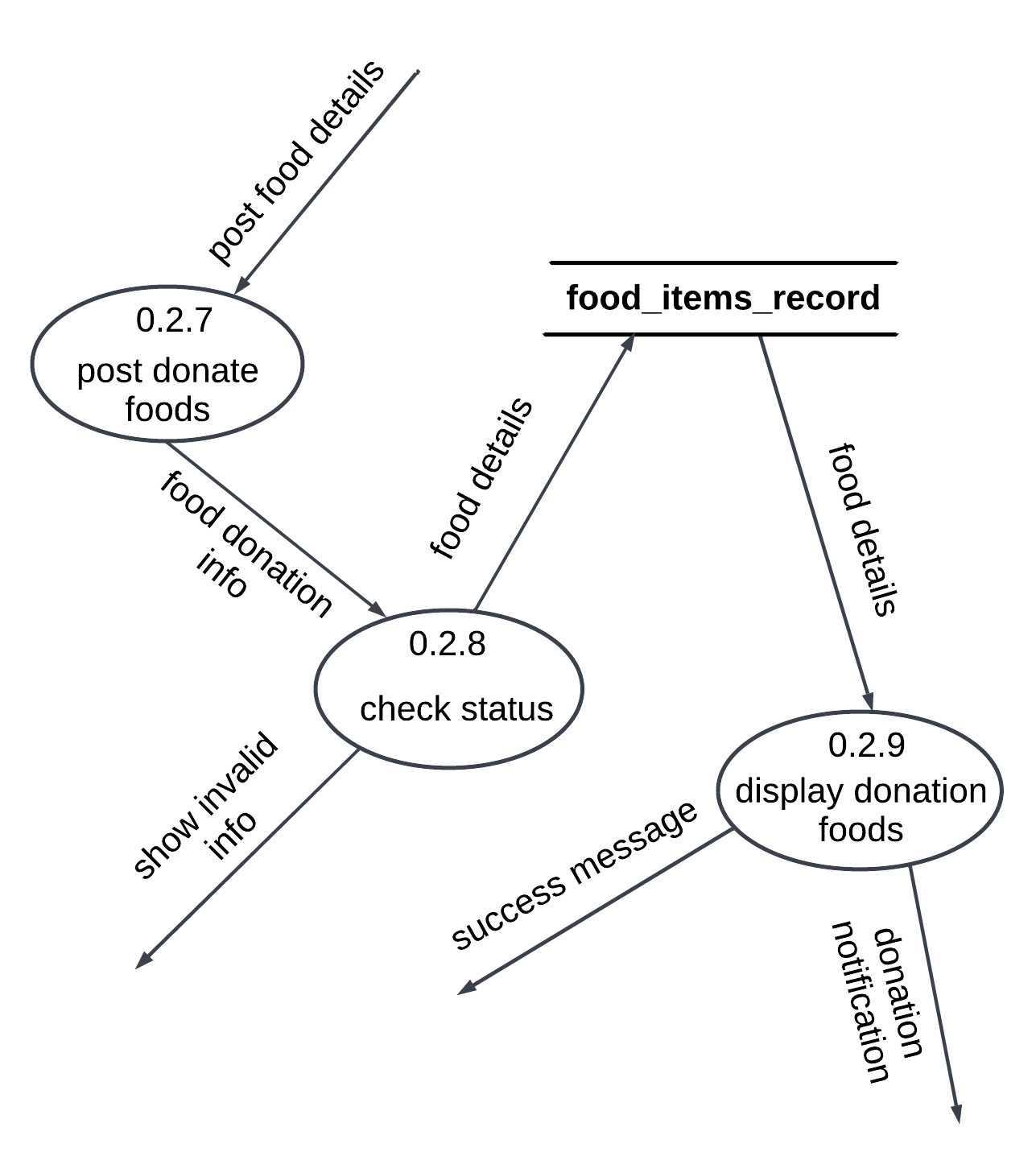


Figure 6: Food donation DFD Level-2

**View Donation History**

A diagram of a food item record

Description automatically generated

Figure 7: View Donation History DFD Level-2

**Donation Rating**

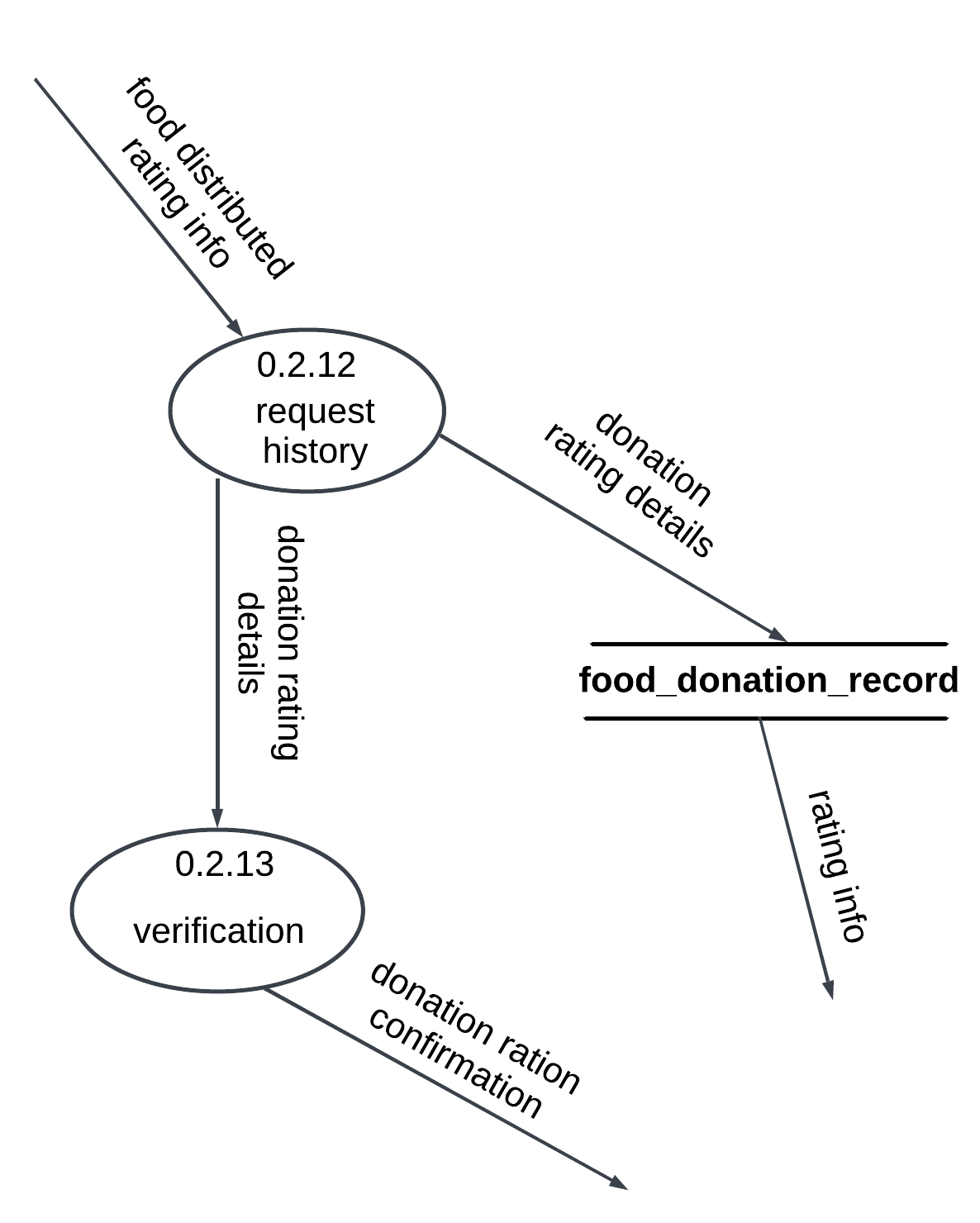


Figure 8: Donation rating DFD Level-2

**Complain with Admin**

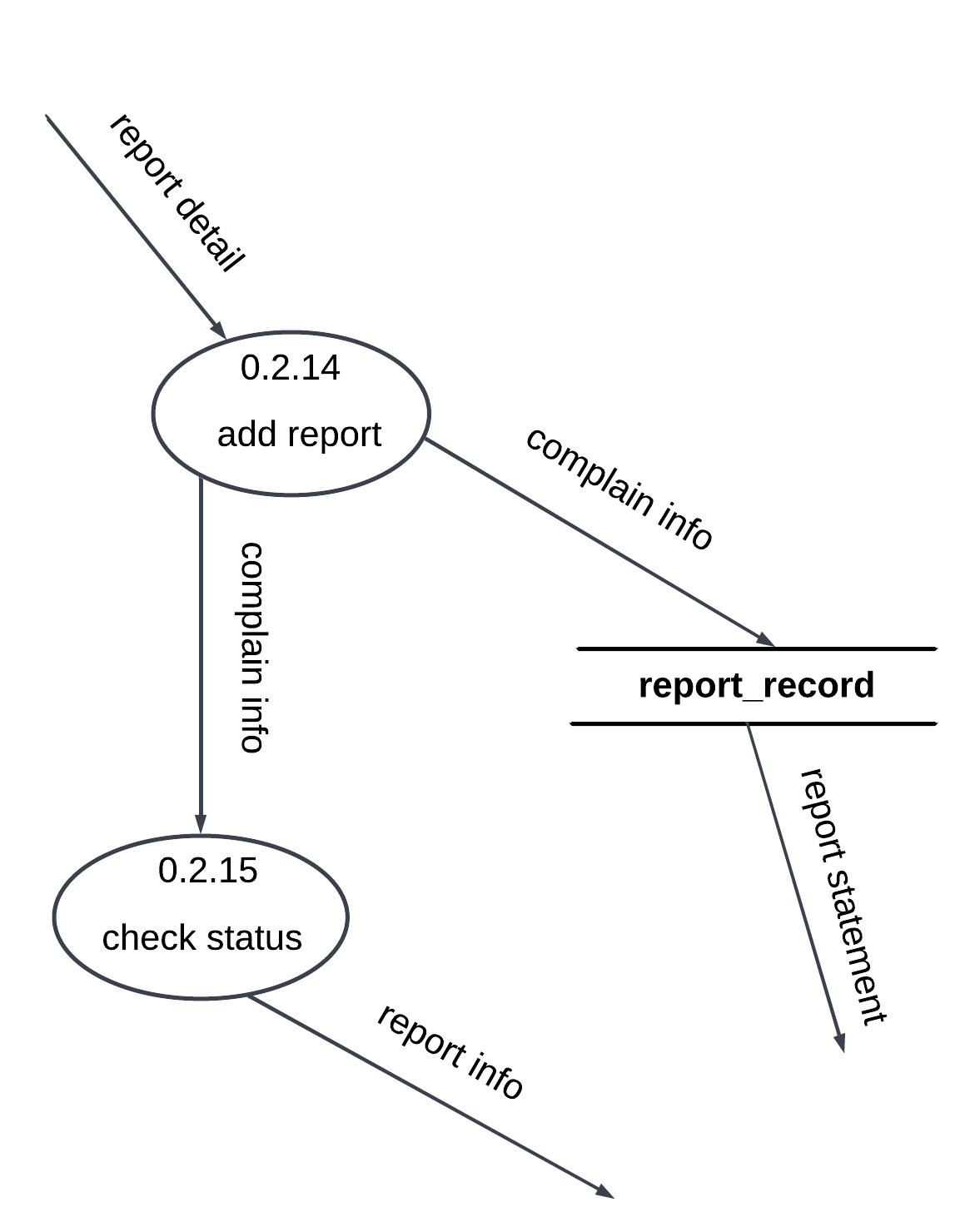
****

Figure 9: Complaint with admin DFD Level-2

**Class Diagram**

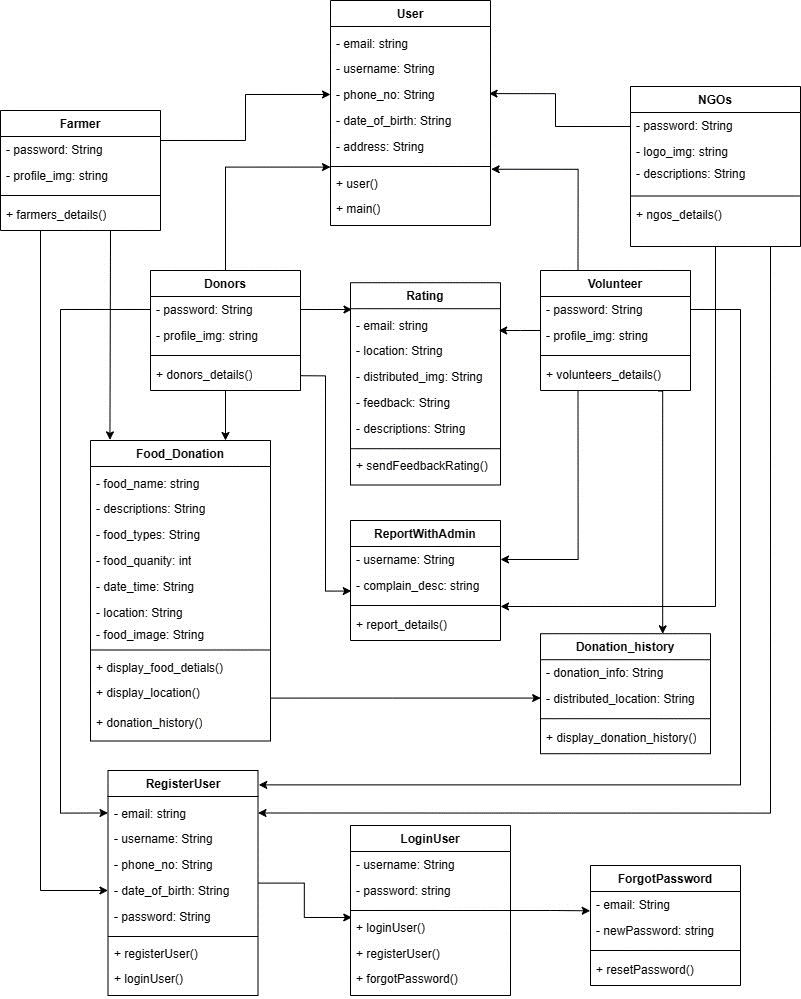


Figure 10: Class diagram of food donation application

**Activity Diagram**

**Login system**

A diagram of a system

Description automatically generated

A diagram of a system

Description automatically generated

Figure 11: Login system activity diagram

**Register System**

A diagram of a system

Description automatically generated

Figure : Register system activity diagram

**Donation System**

A diagram of a system

Description automatically generated

Figure 13: Food donation system activity diagram

**View History**

A diagram of a system

Description automatically generated

Figure 14: View the history system activity diagram

**Donation Rating System**

A diagram of a system

Description automatically generated

Figure 15: Rating donation system activity diagram

**Complaint with Admin**

A diagram of a system

Description automatically generated

Figure 16: Complaint with admin system activity diagram

**Logout System**

A diagram of a user and system

Description automatically generated

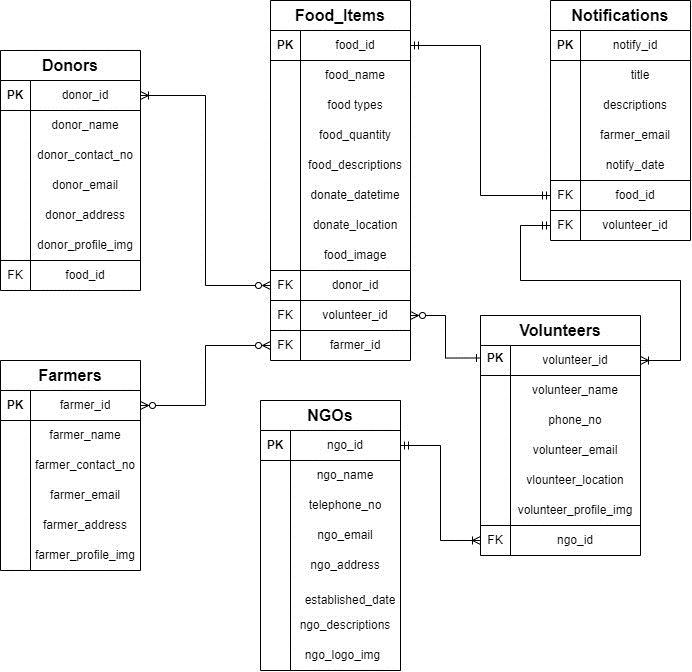
Figure 17: Logout system activity diagram

**Entity Relationship Diagram (ERD)**

A diagram of a farm

Description automatically generated

Figure 18: Initial RED



3 Normalization

Normalization is the process of removing anomalies and reducing data redundancy. Creating tables and supporting these in a manner that is designed to protect the data and increase the database's flexibility by removing duplication and unreliable dependence is necessary. We cannot get the desired outcome when developing entities and their interactions. Anomalies can be added, updated, and removed.

The normalization process for all the attributes is done as shown in the steps given below:

**3.1 Un-Normalization Form (UNF)**

The process of the un-normalization is given below:

* Write all attribute names from the initial ERD with the name of the entry.
* Choose a suitable unique identifier for this entity.
* Show repeating group within {}.
* From the list, attributes are selected as a primary key and it means to be represented with an underline.

Let’s review the un-normalization form of the table again.

This is the list of UNF: -

The NGO ID is the unique identifier of the Un-normalized form so,

**NGOs(**Nog\_id, ngo\_name, telephone\_no, nog\_email, ngo\_address, established\_date, ngo\_descriptions, ngo\_logo\_img, {Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img, {Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_contact\_no, farmer\_email, farmer\_address, farmer\_profile\_img}}**)**

The repeating group is represented inside a curly bracket as shown. We have 3 repeating out of which one is the repeating group.

**3.2 First Normalization Form (1NF)**

The process of the UNF to 1NF is given below:

* The repeating groups should be removed to separate the relations.
* 1NF restriction is built into the relation model.
* The advantage of 1NF is simplicity and uniform access.

All the repeating data are Foods, Donors, and Farmer’s details to remove the all-repeating group.

1. **NGOs (**Nog\_id, ngo\_name, telephone\_no, nog\_email, ngo\_address, established\_date, ngo\_descriptions, ngo\_logo\_img**)**

* **Volunteer\_details (**Ngo\_id\***,** Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img, {Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, food\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_contact\_no, farmer\_email, farmer\_address, farmer\_profile\_img}**)**

The repeating groups should be removed to separate the relations.

1. **Volunteer\_details (**Ngo\_id\***,** Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img**)**
2. **Food\_details(**Ngo\_id\***,** Volunteer\_id\*, Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_contacta\_no, farmer\_email, farmer\_address, farmer\_profile\_img**)**

List of **1NF** tables:

**NGOs (**Nog\_id, ngo\_name, telephone\_no, nog\_email, ngo\_address, established\_date, ngo\_descriptions, ngo\_logo\_img**)**

**Volunteer\_details (**Ngo\_id\***,** Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img**)**

**Food\_details(**Ngo\_id\***,** Volunteer\_id\*, Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_contact\_no, farmer\_email, farmer\_address, farmer\_profile\_img**)**

**3.2 Second Normalization Form (2NF)**

While converting the table to 2NF we should look at the table with a compositive key and review whether each non-key attribute is a dependency on part of the key or all the keys. If the attributes are partially dependent on a composite key, then we must remove the partial key and dependents to a new table.

The process of the 1NF to 2NF is given below:

* The relation is in 2NF.
* The table must be in 1NF.
* Tables should not have a partial dependency.
* All non-key attributes are fully functional and depend on the primary key because it is located within 1NF and not only a part of the primary key.
* A separate Relation should be defined for any attributes that are fully dependent on one attribute of the composite identification.
* Data redundancy is generated by partial functional dependency on an identifier; hence they should be avoided.

Steps to transform into 2NF.

* Identified the functional dependencies in 1NF.
* Each determinant should represent the new relation's primary key.
* Declare as non-key attributes those attributes that are dependent on a specific determinant respecting that determinant.

Identifying Practical Functional Dependency

List every possibility of the composite determinant's primary key and its parts (primary key)

The NGOs table is not a partial dependency, so it is automatically 2NF because this table is not part of the key but the **Volunteers** and **Food\_Details** tables are partial dependencies, so it is normalizing the 2NF.

1. **NGOs (**Nog\_id, ngo\_name, telephone\_no, nog\_email, ngo\_address, established\_date, ngo\_descriptions, ngo\_logo\_img**)**

**Symbolic representations:**

Again, check the **Volunteer\_details** table:

* **Volunteer\_Details (**Ngo\_id\***,** Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img**)**

Ngo\_id\*, Volunteer\_id

Ngo\_id\*

Volunteer\_id volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img

1. **Volunteer\_details** (Ngo\_id\***,** Volunteer\_id\*)
2. **Volunteers** (Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img)

Again, check the **Food \_details** table:

* **Food\_details(**Ngo\_id\***,** Volunteer\_id\*, Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_contact\_no, farmer\_email, farmer\_address, farmer\_profile\_img**)**

Ngo\_id \*, Volunteer\_id\*, Food\_id

Ngo\_id\*, Food\_id

Volunteer\_id\*, Food\_id

Ngo\_id\*, Volunteer\_id\*

Ngo\_id\*

Volunteer\_id\*

Food\_id food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, food\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_contacta\_no, farmer\_email, farmer\_address, farmer\_profile\_img.

1. **Food\_details (**Ngo\_id \*, Volunteer\_id\*, Food\_id\***)**
2. **Foods** **(**Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_contacta\_no, farmer\_email, farmer\_address, farmer\_profile\_img**)**

List of the **2NF** tables:

**NGOs (**Nog\_id, ngo\_name, telephone\_no, nog\_email, ngo\_address, established\_date, ngo\_descriptions, ngo\_logo\_img**)**

**Volunteer\_details** (Ngo\_id\***,** Volunteer\_id\*)

**Volunteers** (Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img)

**Food\_details (**Ngo\_id \*, Volunteer\_id\*, Food\_id\***)**

**Foods** **(**Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_contact\_no, farmer\_email, farmer\_address, farmer\_profile\_img**)**

**3.3 Third Normalization Form (3NF)**

The process of the **2NF** to **3NF** is given below:

Create one relation for each determinant in transitive dependency.

* The table must be in 2NF.
* Inside this new table, a primary key must be selected.
* Identifying dependencies between non-prime key attributes in a table is step one.
* The interdependent non-prime key attributes from the original table should be normalized to create a new table.
* In the original table, this primary key is converted to a foreign key.

The entity has no transactive dependency that is already 3NF. The **Volunteer\_details** and **Food\_details** do have not any non-key attributes, so it has no transactive dependency so automatically 3NF.

1. **Volunteer\_details** (Ngo\_id\***,** Volunteer\_id\*)
2. **Food\_details (**Ngo\_id \*, Volunteer\_id\*, Food\_id\***)**

A transitive dependency happens when two non-primary key attributes are related to one another. Check the transactive dependency of another table because it has more than two or more non-key attributes. The table has more than two non-kye attributes so that checks the transitive dependency as given below:

**NGOs (**Nog\_id, ngo\_name, telephone\_no, nog\_email, ngo\_address, established\_date, ngo\_descriptions, ngo\_logo\_img**)**

**Volunteers** (Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img)

**Foods** (Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_contact\_no, farmer\_email, farmer\_address, farmer\_profile\_img)

Check the transactive dependency:

For **NGOs: -**

* **NGOs (**Nog\_id, ngo\_name, telephone\_no, nog\_email, ngo\_address, established\_date, ngo\_descriptions, ngo\_logo\_img**)**

Ngo\_id ngo\_name

Ngo\_id telephone\_no

Ngo \_id nog\_email

Ngo \_id ngo\_address

Ngo \_id established\_date

Ngo \_id ngo\_descriptions

Ngo \_id ngo\_logo\_img

1. **NGOs (**Nog\_id, ngo\_name, telephone\_no, nog\_email, ngo\_address, established\_date, ngo\_descriptions, ngo\_logo\_img**)**

The NGOs table has no transactive dependency because it was impossible to give the values of other data, so it is also automatically 3NF.

For **Volunteers**: -

* **Volunteers** (Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img)

Volunteer\_id volunteer\_name

Volunteer\_id volunteer\_email

Volunteer\_id phone\_no

Volunteer\_id Duration

Volunteer\_id volunteer\_dob

Volunteer\_id volunteer\_location

Volunteer\_id volunteer\_profile\_img

1. **Volunteers** (Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img)

The Volunteers table has no transactive dependency because it was impossible to give the values of other data, so it is also automatically 3NF. The non-key doesn’t give the non-key attributes.

**Foods** (Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img, Farmer\_id, farmer\_name, farmer\_ contact \_no, farmer\_email, farmer\_address, farmer\_profile\_img)

The Food\_id can be given the Food items detail, Donor\_id and Farmer\_id. So: -

Food\_id food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image,

Then,

Food\_id food\_name

Food\_id food\_ type

Food\_id food\_ quantity

Food\_id food\_ descriptions

Food\_id food\_ datetime

Food\_id donate\_location

Food\_id food\_image

Now,

Food\_id gives the Donor\_id and Farmer\_id can give the Model then Model gives: -

Were,

Food\_id Donor\_id dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img,

Donor\_id dnonr\_name

Donor\_id donor\_contact\_no

Donor\_id donor\_email

Donor\_id donor\_address

Donor\_id donor\_profile\_img

Again,

Food\_id Farmer\_id farmer\_name, farmer\_contact\_no, farmer\_email, farmer\_address, farmer\_profile\_img,

Donor\_id farmer\_name

Donor\_id farmer\_contact\_no

Donor\_id farmer\_email

Donor\_id farmer\_address

Donor\_id farmer\_profile\_img

1. **Foods** **(**Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id\*, Farmer\_id\***)**
2. **Donors** **(**Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img**)**
3. **Farmers (**Farmer\_id, farmer\_name, farmer\_contact\_no, farmer\_email, farmer\_address, farmer\_profile\_img**)**

The Volunteer\_details table is not created because all the attributes are found in the Food\_details table. The final entities and attributes are given below:

List of **3NF**

**Food\_details (**Ngo\_id \*, Volunteer\_id\*, Food\_id\***)**

**NGOs (**Nog\_id, ngo\_name, telephone\_no, nog\_email, ngo\_address, established\_date, ngo\_descriptions, ngo\_logo\_img**)**

**Volunteers** (Volunteer\_id, volunteer\_name, volunteer\_email, phone\_no, volunteer\_dob, volunteer\_location, volunteer\_profile\_img)

**Foods** **(**Food\_id, food\_name, food\_type, food\_quantity, food\_descriptions, donate\_datetime, donate\_location, food\_image, Donor\_id\*, Farmer\_id\***)**

**Donors** **(**Donor\_id, dnonr\_name, donor\_contact\_no, donor\_email, donor\_address, donor\_profile\_img**)**

**Farmers (**Farmer\_id, farmer\_name, farmer\_contact\_no, farmer\_email, farmer\_address, farmer\_profile\_img**)**

A diagram of a food distribution

Description automatically generated with medium confidence

Figure 19: Final ER-Diagram

**System Architecture**

**Wireframe**

**A black and white rectangular object with a x

Description automatically generated**

Figure 20: Splash screen mobile UI

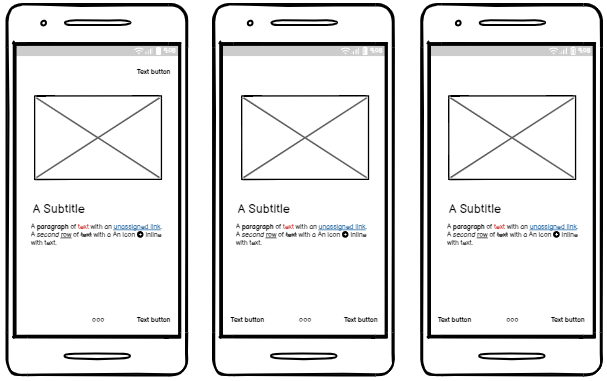


Figure 21: Walkthrough screen mobile UI

A screen shot of a computer

Description automatically generated

Figure 22: Welcome screen mobile UI

A screenshot of a cell phone

Description automatically generated

Figure 23: Login screen mobile UI

A screenshot of a cell phone

Description automatically generated

Figure 24: Register screen mobile UI

A screenshot of a phone

Description automatically generated

Figure 25: Register response message mobile UI

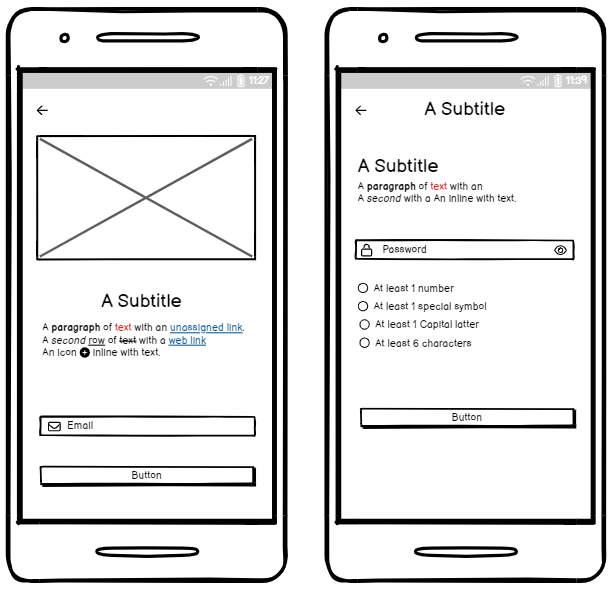


Figure 26: Forgot password screen mobile UI

A screenshot of a computer screen

Description automatically generated

Figure 27: Home screen mobile UI

A screen shot of a phone

Description automatically generated

Figure 28: History screen mobile UI

A screenshot of a computer

Description automatically generated

Figure 29: Post/Donation screen mobile UI

A screenshot of a mobile phone

Description automatically generated

Figure 30: Profile screen mobile UI

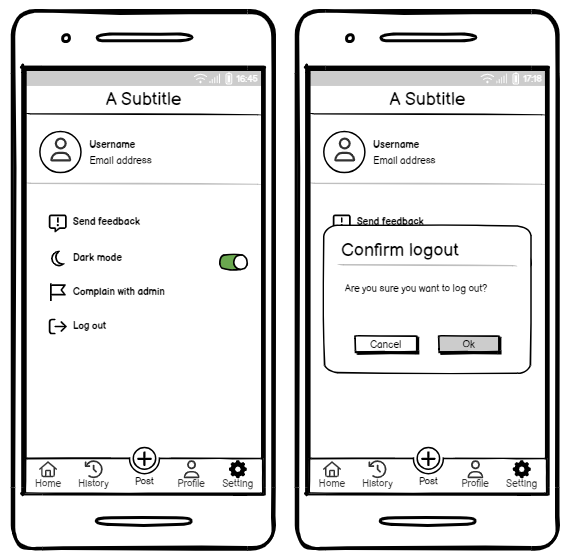


Figure 31: Setting screen mobile UI