FARMO DEVELOPER NOTES

Types of User: Inside the system

1. Admin
2. SuperAdmin
3. Farmer
4. VerifiedFarmer
5. Consumer

Types of User: Outside the system

Means this is for webpage, app which show to user.

1. Admin **]Admin = Admin | Admin = Super Admin[**
2. Farmer **]Farmer = Farmer | Farmer = Verified Farmer[**
3. Consumer

Connection Method: Between Server and User

POST

GET

Profile Status:

profile_status = [

- PENDING – the profile is created but not yet reviewed or approved.
- ACTIVE – the profile is verified and currently in use.
- SUSPENDED – the profile is temporarily disabled due to issues or violations.
- DELETED – the profile has been removed or deactivated permanently

]

Possible Activity Types

THIS ACTIVITIES ARE USED TO TRACK THE USER ACTIVITIES. [USED IN BACKEND]

- **Authentication & Account**
  - LOGIN
  - LOGOUT
  - PASSWORD_CHANGE
  - UPDATE_PROFILE_PIC
  - PROFILE_UPDATE
  - VERIFICATION_SUBMITTED
  - VERIFICATION_APPROVED
  - VERIFICATION_REJECTED

- **Wallet & Transactions**
  - WALLET_CREATED
  - WALLET_PIN_CHANGED
  - WALLET_TOPUP
  - WALLET_DEBIT
  - TRANSACTION_INITIATED
  - TRANSACTION_SUCCESS
  - TRANSACTION_FAILED

- **Products**
  - PRODUCT_CREATED
  - PRODUCT_UPDATED
  - PRODUCT_DELETED

- **Orders**

    - ORDER_PLACED
    - ORDER_UPDATED
    - ORDER_CANCELLED
    - ORDER_FULFILLED

- **Ratings**
    - RATE_PRODUCT
    - RATE_FARMER
    - FARMER_RATING_UPDATE
    - PRODUCT_RATING_UPDATE
    - FARMER_RATING_DELETE
    - PRODUCT_RATING_DELETE

- **General Activity**
    - SEARCH
    - VIEW_PROFILE
    - ACTIVITY_LOGGED - For other activity.

Important Point to Remember:

Use communication Terms to communicate between backend and Frontend as it is. No change in case. Like do not use admin, farmer, pending and Login instead of Admin, Farmer, PENDING and login.

Communications Terms: User Types, profile status, Error code, terms inside "…", etc.

Examples of Terms:

Admin, SuperAdmin, Farmer, VerifiedFarmer, PENDING, ACTIVE, DELETED, login_with_token, login, register, user_id, is_admin, ACCOUNT_INACTIVE, INVALID_CREDENTIALS, login_access.

[**Note**: To separate all terms from others, they are written in "Green Letters" in this doc. But "Green Letters" itself is not a term.]

Request Handling:

Request are generally handled with these HTTP codes. We see some where during browsing 404 is not found like that.

- **200 OK** → success, existing resource returned. All good
- **201 Created** → success, new resource created like data saved.
- **401 Unauthorized** → token invalid/missing.
- **403 Forbidden** → authenticated but not allowed.
- **400 Bad Request** → invalid input.
- **500 Internal Server Error** → backend failure.

Suggested Way: eutei thau ekei choti vaya ghari ghari lekhnu pardai na.

Pseudo Code:

File.java

```
Import errorHandler;

ER = errorHandler(); // creating object

function requestToServer(url, method, data, token):
    response = sendRequest(url, method, data, token)

    if response.code == 200 or response.code == 201:
        // ✅ Success
       // handleSuccessful request according to request type
    else:
        // ❌ Error
        ER.error_handler(response)
```

---

errorHandler.java

```
class errorHandler{
        function error_handler(response):
            switch(response.code):
                case 400:
                    // or popup dialog box dekhai dinu
                    print("Invalid input: " + response.error)
                case 401:
                    print("Unauthorized: " + response.error)
                case 403:
                    print("Forbidden: " + response.error)
```

```
            case 404:
                print("Not Found: " + response.error)
            case 500:
                print("Server Error: " + response.error)
            default:
                print("Unexpected Error: " + response.code + " " + response.error)
}
```

Login Mechanism: use POST Method

Path for “login_with_token” : 'https://……/api/auth/login-with-token/'

Path for “login” : 'https://……/api/auth/login/'

Step in login,

1. takes userID or Phone and Password from **user**.
2. Also collect extra two type of data internally in system i.e**. is_admin** and **device_info**
3. Send to server.
4. Server search user in database, if matches; server create token, refresh_token for login.
5. Server send that token with refresh_token and UserID back to User device.

User uses that token and UserID for any other request like for login, access something.

User to Server: Data flow [request]

First Login:

For Admin [Web]: Admin login every time.

{

“identifier” = userID or Phone [string]

“password” = password [String]

“is_admin” = True [Boolean]

“device_info” = Windows 11 Brave [String]

}

For Farmer/Comsumer[App]:

{

“identifier” = userID or Phone [string]

“password” = password [String]

“is_admin” = False [boolean]

“device_info” = Samsung A52 [String]

}

Second Login: Only after “remember me” checked.

For Both:

{

“user_id” = xxxxxx [String]

“token” = xxxxxxx [String]

“refresh_token” = xxxxxxxx [String]

“is_admin” = True or False [boolean]

“device_info” = Samsung A52 [String]

}

Server check “token” and “UserID” and give access.

Sever to User: [respond]

Each time “First Login”

For Both:

{

“token” = xxxxxx [String]

“refresh_token” = xxxxxxxx [String]

“user_id” = xxxxxx [String]

}

Save those data[token, refresh_token and userID] in device.

Second Time Login if user checked remember me.

{

“token” = token or none

“refresh_token” = refresh-token or none

"user_id" = userID or none

}

If login token is expired:

[For User Expiry time: 40 days && For Admin Expiry time: 12 hrs]

Generally, user send "token" and "userID" for login.

If token is expired. Server uses "refresh_token" to create new "token" and send back to user.

{

"token" = xxxxxx [String]

"refresh_token" = xxxxxxxx [String]

"user_id" = xxxxxx [String]

}

Error Type:

Type of Error [error_code]:

- MISSING_CREDENTIALS - Missing identifier or password
- INVALID_CREDENTIALS - Wrong password
- ACCOUNT_PENDING - Account awaiting activation
- ACCOUNT_INACTIVE - Account blocked/suspended
- MISSING_TOKENS_OR_USERID – missing token or refresh token or user_id
- INVALID_TOKEN - Token expired or invalid

Server to User: For Login failed [respond]

{

"req_access" = False [Boolean]

"error_code" = Code Word [String]

}

Example: For Wrong Password

{

“req_access” = False

“error_code” = “INVALID_CREDENTIALS”

}

For Account Pending:

It occurs only when Admin creates new user, then admin put any password and make profile Status pending.

To activate account, User [both types of user] have to login with that password given by admin and they have interface to change password with confirm password.

After changing password, account have activated and they again redirected in Login Page.

Then process goes to login.

Signup Mechanism: use both POST and GET Method

Path for “register” : “https://...../”

Fields:

1. userID [Required]
2. Password [Required]
3. First Name [Required]
4. Middle Name
5. Last Name [Required]
6. Sex [Required]
7. Date of Birth [Required]
8. User Type [Required]
9. Province [Required]
10. District [Required]
11. Municipal [Required]
12. Ward [Required]
13. Tole [Required]
14. Phone [Required]
15. Secondary Phone
16. Email
17. Facebook
18. WhatsApp
19. About
20. Profile Picture

Data Send To Server: [request]

{

“user_id” : userID

“password” : Password

“f_name” : First Name

“m_name” : Middle Name

“l_name” : Last Name

“sex” : Sex

“dob” : Date of Birth

“user_type” : User Type

“province” : Province

“district” : District

“municipal” : Municipal

“ward” : Ward

“tole” : Tole

“phone” : Phone

“phone2” : Secondary Phone

“email” : Email

“facebook” : Facebook

“whatsapp” : WhatsApp

“about” : About

“profile_pic” : Profile Picture

“created_by” : Created_By

}

Server Response to Web/App:

For Registration Faild:

{

'req_access': False [Boolean]

'error_code': 'MISSING_REQUIRED_FIELDS' [String]

}

For Registration Success:

{

'req_access': True

}

For UserID:

Path for 'check_userid' : 'https://……./api/auth/check-userid/',

UserID needs to check constantly, entered userID is available or not.

If userID textbox is empty then nothing to worry about.

If a single or more characters are present, then box sent request to server to checked userID is available or not.

Request is to be sent in change in every character in textbox.

Request: App/Web to Server

{

"user_id" : UserID

}

Response: Server to App/Web

{

"status" : 1 or 0 [Integer]

}

Status Int mean:

1 → For Not Available

0 → For Available

For User Type:

For Web:

user_type = [SuperAdmin, Admin, Farmer, Consumer]

For App:

user_type = [Farmer, Consumer]

For Created_By: [PRIVILEGES]

Values : [

SuperAdmin – Access to create super-admin, admin, farmer and consumer [***Web***]

Admin – Access to create farmer and consumer [***Web***]

Itself – farmer and consumer create their account itself. [***App***]

]

In Web four items for Created By [Super Admin, Admin, Farmer and Consumer]

In App two items for Created By [Farmer and Consumer]

Error Handling in Signup/Register:

Type of Error [error_code]:

- **MISSING_REQUIRED_FIELDS** – Missing mandatory signup fields
- **USERID_EXISTS** – User ID already exists
- **INSUFFICIENT_PRIVILEGES** – Signup requires higher privileges. Means Admin do not allowed to created super-admin
- **PHONE_NUMBER_ACCOUNT_LIMIT_REACHED** – Phone number reached account limit. One phone number allowed to create maximum 3 accounts.
- **PHONE_EXISTS_ACTIVE_ACCOUNT** – Phone already linked to active account. 1 phone number allowed to be have 1 ACTIVE account.
- **PROFILE_PICTURE_UPLOAD_FAILED**

Example:

{

'req_access': False

'error_code': `**PHONE_EXISTS_ACTIVE_ACCOUNT**`

}