Trust-Based Lending Manager – API Guidelines

1. Purpose

This API is designed to support **informal lending between friends and family** while preserving trust, transparency, and accountability. It intentionally avoids legal, coercive, or enforcement-driven language and behavior.

The system acts as a **shared memory and gentle facilitator**, not a debt collection tool.

---

2. Core Principles

2.1 Trust First

- No penalties, interest, or legal enforcement
- All data is mutually visible to involved participants
- Changes require acknowledgment or consent

2.2 Transparency

- Every action is logged and visible
- Both parties see the same balance and history
- Reasons can be attached to delays or changes

2.3 Soft Accountability

- Gentle reminders instead of demands
- Borrowers can snooze reminders with explanations
- Payments can be recorded by either party but require confirmation

2.4 Human Language

Avoid harsh financial terminology: | Avoid | Prefer | |------|--------| | Loan | Arrangement | | Debtor | Borrower | | Due Date | Expected By | | Penalty | ❌Not supported |

---

3. API Conventions

3.1 Authentication Overview

The API uses **token-based authentication** (JWT).

Flow: 1. User signs up or logs in 2. Server issues an access token 3. Client sends token in `Authorization` header for all protected APIs

```
Authorization: Bearer <access_token>
```

All arrangement, payment, reminder, and activity endpoints **require authentication**.

3.2 Base URL

```
/api/v1
```

3.3 Roles

Each arrangement has exactly two roles: - `lender` - `borrower`

Roles affect permissions but **never restrict visibility**.

3.1 Base URL

```
/api/v1
```

3.2 Authentication

- Token-based (JWT / Session)
- All endpoints require authentication unless stated otherwise

3.3 Roles

Each arrangement has exactly two roles: - `lender` - `borrower`

Roles affect permissions but **never restrict visibility**.

---

4. Authentication APIs

Authentication is intentionally simple and friendly. No KYC, no identity verification beyond email.

4.1 Sign Up

```
POST /auth/signup
```

Request:

```
{
  "name": "Bhavya",
  "email": "bhavya@email.com",
```

```
  "password": "strong-password"
}
```

Server Response (201 Created):

```
{
  "userId": "u_123",
  "accessToken": "jwt-token",
  "message": "Account created successfully"
}
```

4.2 Login

```
POST /auth/login
```

Request:

```
{
  "email": "bhavya@email.com",
  "password": "strong-password"
}
```

Server Response (200 OK):

```
{
  "userId": "u_123",
  "accessToken": "jwt-token",
  "message": "Welcome back"
}
```

4.3 Logout

```
POST /auth/logout
```

Request: *No body*

Server Response (200 OK):

```
{
  "message": "Logged out successfully"
}
```

---

5. Core Resources

5.1 User

Represents a real person using the system.

Key fields: - `id` - `name` - `email` - `timezone`

4.1 User

Represents a real person using the system.

Key fields: - `id` - `name` - `email` - `timezone`

4.2 Arrangement

Represents a trust-based lending agreement.

Key fields: - `id` - `title` - `totalAmount` - `currency` - `expectedBy` - `repaymentStyle` (`one_time`, `installments`, `flexible`) - `note` - `status` (`pending`, `active`, `closed`)

An arrangement becomes active only after both participants confirm.

---

5. API Endpoints

All endpoints follow a **request → server processing → response** model. Below, each endpoint explicitly documents: - Data sent by the client (request body) - Data returned by the server (response body)

HTTP status codes are used meaningfully (200, 201, 400, 403, 404).

5.1 Create Arrangement

```
POST /arrangements
```

Request (from lender):

```
{
  "title": "Laptop help",
  "totalAmount": 45000,
  "currency": "INR",
  "borrowerEmail": "friend@email.com",
  "expectedBy": "2026-06-01",
  "repaymentStyle": "flexible",
  "note": "Pay back when comfortable, no pressure ❤️"
}
```

Server Response (201 Created):

```
{
  "id": "arr_456",
  "status": "pending",
  "message": "Invitation sent to borrower",
  "createdAt": "2026-01-27T10:30:00Z"
}
```

5.2 View Arrangements

```
GET /arrangements
```

Request: *No body*

Server Response (200 OK):

```
[
  {
    "id": "arr_456",
    "title": "Laptop help",
    "role": "lender",
    "totalAmount": 45000,
    "balanceRemaining": 12000,
    "status": "active"
  }
]
```

5.3 Arrangement Details

```
GET /arrangements/{id}
```

Request: *No body*

Server Response (200 OK):

```
{
  "id": "arr_456",
  "title": "Laptop help",
  "totalAmount": 45000,
  "paidAmount": 33000,
  "balanceRemaining": 12000,
  "expectedBy": "2026-06-01",
  "repaymentStyle": "flexible",
  "participants": [
    { "userId": "u_123", "role": "lender" },
    { "userId": "u_789", "role": "borrower" }
  ],
  "note": "Pay back when comfortable ❤️",
  "status": "active"
}
```

---

6. Payments

6.1 Record Payment

```
POST /arrangements/{id}/payments
```

Request (from either party):

```
{
  "amount": 5000,
  "paidOn": "2026-01-20",
  "note": "Sent via UPI"
}
```

Server Response (201 Created):

```
{
  "paymentId": "pay_901",
  "status": "pending_confirmation",
  "recordedBy": "borrower",
  "message": "Payment recorded and awaiting confirmation"
}
```

6.2 Confirm Payment

```
POST /payments/{paymentId}/confirm
```

Request (from lender):

```
{
  "confirmed": true
}
```

Server Response (200 OK):

```
{
  "paymentId": "pay_901",
  "status": "confirmed",
  "balanceRemaining": 12000,
  "message": "Payment confirmed successfully"
}
```

---

7. Reminders

7.1 Create Reminder

```
POST /arrangements/{id}/reminders
```

Request (from lender):

```
{
  "schedule": "monthly",
  "messageTone": "gentle",
  "customMessage": "Hey! Just a small reminder — no rush at all 📶"
}
```

Server Response (201 Created):

```
{
  "reminderId": "rem_301",
  "nextTrigger": "2026-02-01",
  "status": "active"
}
```

7.2 Snooze Reminder

```
POST /reminders/{id}/snooze
```

Request (from borrower):

```
{
  "snoozeUntil": "2026-03-01",
  "reason": "Salary delayed this month"
}
```

Server Response (200 OK):

```
{
  "reminderId": "rem_301",
  "status": "snoozed",
  "visibleNote": "Salary delayed this month"
}
```

---

8. Proposals & Changes

Used for modifying expectations collaboratively.

8.1 Propose Change

```
POST /arrangements/{id}/proposals
```

Request:

```
{
  "type": "expectedByChange",
  "newExpectedBy": "2026-08-01",
  "reason": "Job switch took longer than expected"
}
```

Server Response (201 Created):

```
{
  "proposalId": "prop_551",
  "status": "pending",
```

```
  "message": "Proposal sent for review"
}
```

8.2 Respond to Proposal

```
POST /proposals/{proposalId}/respond
```

Request:

```
{
  "decision": "accept"
}
```

Server Response (200 OK):

```
{
  "proposalId": "prop_551",
  "status": "accepted",
  "updatedExpectedBy": "2026-08-01"
}
```

---

9. Activity Log

9.1 View Activity

```
GET /arrangements/{id}/activity
```

Request: *No body*

Server Response (200 OK):

```
[
  {
    "type": "payment_confirmed",
    "by": "lender",
    "message": "₹5000 confirmed",
    "timestamp": "2026-01-20T10:45:00Z"
  }
]
```

---

10. Trust Summary (Non-Financial)

10.1 View Trust Summary

```
GET /arrangements/{id}/trust-summary
```

Request: *No body*

Server Response (200 OK):

```
{
  "paymentsOnTimeRatio": 0.83,
  "communicationScore": "good",
  "lastInteraction": "2026-01-21",
  "summary": "Healthy and communicative arrangement"
}
```

---

11. Closing an Arrangement

11.1 Close Arrangement

```
POST /arrangements/{id}/close
```

Request:

```
{
  "message": "All settled. Thanks for the trust   "
}
```

Server Response (200 OK):

```
{
  "status": "closed",
  "closedAt": "2026-01-27T12:00:00Z"
}
```

---

12. Non-Goals (Explicitly Unsupported)

The API **does not support**: - Interest calculation - Penalties or late fees - Legal contracts or enforcement - Credit scoring - Public reputation systems

13. Design Philosophy Summary

This API exists to: - Preserve relationships - Reduce awkward conversations - Encourage honesty and communication - Act as a neutral, shared record

Trust is the feature.

End of Document