FinTrustChain

Introduction

FinTrustChain is an app for decentralized unsecured loans based on community trust. We mimic the credit score system using our TrustIndex metric and have incorporated an endorsement system that leverages community connections. This platform enables users to quickly access loans without traditional collateral, relying instead on social trust and reputation within the community.

Formulas:

1. Receiver:

TrustIndex Gain

$$TIGain = b + t + a$$

where,

$$baseGain(b) = alpha * (950 - TI)/4$$

$$timelinessBonus(t) = (days_early/repayment_days) * 2$$

$$amountFactor(a) = 1 + (Loan_amount/40000)$$

TrustIndex Loss

$$TILoss = b + t + a$$

where,

$$baseLoss(b) = beta * (950 - TI)/2$$

$$timelinessBonus(t) = (days_late/7) * 2$$

$$amountFactor(a) = 1 + (Loan_amount/20000)$$

Alpha-Beta Values:

TI Range	Alpha	Beta
below 400	0.05	0.01
400-500	0.15	0.02
500-600	0.12	0.05
————————	————————	————————
600-750	0.1	0.08
750-850	0.08	0.1
850-900	0.05	0.12
900-950	0.02	0.15

2. Endorser:

$$TIGain = (RecTIGain)/(No.OfEndorserOfReceiver)$$

$$TILoss = (RecTILoss)/(No.OfEndorserOfReceiver)$$

3. Guarantor :

$$TIGain = (RecTIGain)/2$$

$$TILoss = (RecTILoss)/2$$

Initial TI gain through endorsement and loss:

When the user get endorsed he will get some TI value added.

And if someone removes the endorsement there will be loss of TI.

TI Range	TI gain	TI loss
below 400	20	5
400-500	20	5
500-600	15	8
600-750	10	10
750-850	8	15
850-900	5	20
900-950	2	30

Workflow:

1. Endorsement

- You can only give endorsement if you know the User ID of person (you have to search through ID and then endorse him)

1. Initial endorsement gain and endorsement removal loss (as explained in table)
2. Endorser will get TIGain and Loss as per above formulas: click here
3. Endorser will get connected bidirectionally (If you endorse someone he will become your endorser too)
4. Every month only four endorser can be connected.

2. Guarantor flow:

- If endorser is unable to pay the loan in first 28 days of default loan will be transferred to Guarantor

3. Loan Workflow :

Flow 1 : Loan Process

1. Receiver will see the list of lender's brochures

   (this list will be shown as per receiver criteria e.g. TI)

   - Lenders will post the Loan brochures that will contain
     1. Loan amount
     2. Interest Rate
     3. Time/days of repayment
     4. Historic data : (No. of loan given, req/accept ratio, amount of loan lent)
2. Receiver can select up to 3 lenders at a time.

   (Whichever lender accept first that lender will get connected and request will expire for other lenders : Simply FCFS)
3. If lender accept request then there will be standard contract.
   - Contract is 3 pages pdf each will be the agreement by each party. and those are
     1. Receiver: It will be the acceptance of Lenders agreement and standard agreement
     2. Guarantor: Agreement if default
     3. Lender: Acceptance of request
   - When each accepts and tick the agreement box the e-signature will be signed.
   - Each Contract will have their own unique Id
4. Payment will be initiated.
5. As soon as the lender sends money loan repayment count down will start
6. Loan Repayment will be in installment.

- Loan repayment acknowledgement letter : Will correspond to Contract. (Auto signed after full repayment by all 3) - Lender, Receiver and Guarantor will receive the letter.

Flow 2 : Contract

1. If guarantor don’t accept and sign contract then receiver has to another guarantor.
2. If Lender or Receiver cancels the contract then entire contract will be cancelled.

- Receiver → Guarantor → Lender
- There is Common template pdf (unsigned contract template) : Copy of this pdf is made on every contract.
    - When Each one of entity (i.e. Lender, Receiver and Guarantor) signs the copy will be replaced by signed contract. If any one rejects contract get deleted.

Extra Details :

- Every user has 2 profiles - Lender and Receiver (you can toggle between them)
- If you have ongoing loan as receiver then you can’t lend the money as lender and vice versa (One role at a time)
- While login we have to save the png for e-sign. (while login there will be section in form)

User Dashboard

1. Receiver

i. Identity and trust Section (public) :

- Basic info and

1. Breakdown of trustIndex
2. Verification badge
3. No. of defaults (if any)
4. Pan Verification state (Future scope)

ii. Loan Activity (Private) :

1. My Active loan ：Amount, Lender name and Repayment & status

2. Loan Request history : Approve-reject-pending
3. Repayment schedule
4. Past Contracts.

iii. Guarantor and endorser Activity (public + private):

1. Endorsed User List (With current trust Index) - Public
2. Inbox : Request from other to be Endorser (Accept/Decline) - Private
3. Inbox : Request from other to be guarantor (Accept/Decline) - Private
4. Active guarantor responsibility - Private
5. Past guarantor responsibility - Private

iv. Trust Index Analytics (Private) :

1. Endorsement
2. Guarantor

v. Milestone (Public) :

- Shows the eligibility for greater loan

TI vs Eligible loan amount:

TI Range	Laon Amount (INR)
below 400	500
400-500	1000
500-600	2000
600-700	5000
750-850	10000
850-900	15000
900-950	20000

2. Lender

i. Portfolio Overview (Public + Private)

1. Active Loan Issued (count + amount) - Public
2. Average Interest Rate - Public
3. Average Repayment Time - Public

4. Default rate/ Interest rate - Public
5. Current active loan details - Private

ii. Loan History table and Loan Request inbox (Private) :

1. Borrower name
2. Borrower Amount
3. Purpose, trustIndex, return and action (accept/reject)
4. Loan History - Contracts and acknowledgement letter
5. Inbox :
    - Actions - View Profile, lend (accept) or decline
    - View Profile Details :
        a. trustIndex
        b. Endorsement List
        c. Repay Record
        d. Pan verification badge (Future scope)

iii. Transaction Ledger (Payment history - send/receive)

Project Stack:

1. Backend : Node.js + Express.js
2. Authentication : JWT token. (Email Authentication by nodemailer)
3. Frontend : React
4. Database : MongoDB (atlas)
5. Storage : Multer (To save the e-sign and contract)
6. Payment Portal - Razor pay mock / Slice mock.

- Use PDF.js for pdf view and pdf-lib for pasting the sign onto the pdf.

FinTrustChain: Key Backend Design — Endpoints & MongoDB Schemas

1. High-Level REST (JSON) API Map

Group	Method & Path	Purpose	Auth
Auth	POST `/auth/register`	Create user (captures e-sign PNG, email verification mail)	—
	POST `/auth/login`	Return JWT	—
	POST `/auth/verify-email`	Verify token sent by nodemailer	—
Profile	GET `/users/:id/public`	Public profile (TI breakdown, badges, milestones, endorsements)	public
	GET `/users/:id/private`	Owner's private view (loan activity, inboxes, analytics)	JWT
	PATCH `/users/:id`	Update limited fields (avatar, bio)	JWT
Role Toggle	POST `/users/:id/toggle-role`	Switch between Lender and Receiver if no active opposite role	JWT
Endorsements	POST `/endorsements`	Create endorsement (body: receiverId)	JWT
	DELETE `/endorsements/:id`	Remove endorsement	JWT
Guarantor Requests	POST `/guarantor-requests`	Ask someone to be guarantor	JWT
	PATCH `/guarantor-requests/:id`	Accept/decline	JWT
Loan Brochures (Lender-side)	POST `/brochures`	Post lending offer (amount, rate, tenor)	JWT
	GET `/brochures`	Paginated public list filtered by requester TI	JWT optional
	PATCH `/brochures/:id`	Edit/close offer	JWT
Loan Requests (Receiver-side)	POST `/loan-requests`	Select up to three brochure IDs	JWT
Contracts	POST `/contracts/:loanRequestId/sign`	Sign as receiver/guarantor/lender	JWT

Group	Method & Path	Purpose	Auth
	DELETE `/contracts/:id`	Cancel (before all parties sign)	JWT
Payments	POST `/payments/:contractId/installment`	Record repayment webhook/trigger TI gain	JWT
Analytics	GET `/trust-index/history`	Time-series for dashboards	JWT
Files	GET `/files/:id`	Signed contract PDF or e-sign PNG via GridFS/Multer	JWT (owner+counterparts)

2. Core MongoDB Schemas

2.1 User

```
{
  _id: ObjectId,
  email, passwordHash,
  name, avatarUrl,
  currentRole: { type: String, enum: ['LENDER','RECEIVER'], default:'RECEIVER' },
  trustIndex: Number,                    // 0-950
  trustBreakdown: { base: Number, timeliness: Number, amount: Number }, // cached
  verification: {
    emailVerified: Boolean,
    panVerified: Boolean                 // future scope
  },
  eSign: { fileId: ObjectId, filename },
  milestones: { eligibleLoan: Number },

  // relationships
  endorsementsGiven: [{ type:ObjectId, ref:'User' }],
  endorsementsReceived: [{ type:ObjectId, ref:'User' }],

  // analytics
  tiHistory: [{ value:Number, date:Date }],

  createdAt, updatedAt
}
```

2.2 Endorsement

```
{
  _id: ObjectId,
  endorser: { type:ObjectId, ref:'User' },
  receiver: { type:ObjectId, ref:'User' },
  createdAt,
  removedAt
}
```

2.3 GuarantorRequest

```
{
  _id: ObjectId,
  receiver: { type:ObjectId, ref:'User' },
  guarantor: { type:ObjectId, ref:'User' },
  loanRequest: { type:ObjectId, ref:'LoanRequest' },
  status: { type:String, enum:['PENDING','ACCEPTED','DECLINED'] },
  createdAt, updatedAt
}
```

2.4 LoanBrochure (Lender Offer)

```
{
  _id: ObjectId,
  lender: { type:ObjectId, ref:'User' },
  amount: Number,
  interestRate: Number,      // APR %
  tenorDays: Number,
  stats: {
    loansGiven: Number,
    reqAcceptRatio: Number,
    totalLent: Number
  },
  active: Boolean,
  createdAt, updatedAt
}
```

2.5 LoanRequest

```
{
  _id: ObjectId,
```

```
  receiver: { type:ObjectId, ref:'User' },
  brochureIds: [{ type:ObjectId, ref:'LoanBrochure' }],
  selectedBrochure: ObjectId,  // set on first-come-first-served acceptance
  status: { type:String, enum:['PENDING','PARTIALLY_FILLED','CONTRACTING','CANC
ELLED'] },
  createdAt
}
```

2.6 Contract

```
{
  _id: ObjectId,
  contractId: String,                // unique 3-way pdf id
  loanRequest: { type:ObjectId, ref:'LoanRequest' },
  lender, receiver, guarantor: { type:ObjectId, ref:'User' },
  principal: Number, interestRate: Number, tenorDays: Number,
  pdfFileId: ObjectId,               // signed pdf in GridFS
  signatures: {
    receiver: Boolean,
    guarantor: Boolean,
    lender: Boolean
  },
  status: { type:String, enum:['PENDING','ACTIVE','CANCELLED','REPAID','DEFAULT','G
UARANTOR_PHASE'] },
  repaymentSchedule: [{
    dueDate: Date,
    amountDue: Number,
    paid: Boolean,
    paidDate: Date
  }],
  createdAt, updatedAt
}
```

2.7 Payment

```
{
  _id: ObjectId,
  contract: { type:ObjectId, ref:'Contract' },
  payer: { type:ObjectId, ref:'User' },
  amount: Number,
  razorpayRef: String,
```

```
  installmentIndex: Number,
  onTime: Boolean,
  createdAt
}
```

3. Crucial Field Notes

- **trustIndex** is recalculated in a service layer whenever an event occurs (endorsement, repayment, default) using the Alpha/Beta tables and formulas.
- **tiHistory** lets you plot trendlines in Trust Index Analytics.
- **signatures** tri-states allow the PDF to be overwritten by pdf-lib each time a new party signs.
- **status transitions** in Contract enable automatic move to GUARANTOR_PHASE after 28 days of default.
- **endorsements: Given/Received** allow the “only four endorsers per month” rule—enforce with a MongoDB aggregation on creation.
- **currentRole** guard middleware blocks lending while a borrow contract is ACTIVE (and vice versa).
- **repaymentSchedule** is generated on contract activation; each Payment webhook marks installments as paid and triggers TI gain for receiver/endorser.

4. Suggested Folder Structure

```
server/
  ├─ controllers/
  ├─ routes/
  ├─ models/
  ├─ services/
  │    ├─ trustIndex.service.js
  │    └─ pdf.service.js
  ├─ middlewares/
  ├─ utils/
  └─ config/
```

5. Next Steps (“More...”)

1. **Services**
   - Implement `trustIndex.service.js` to centralize all gain/loss math.

- Add `loanMatcher.service.js` for FCFS lender acceptance.

2. **Cron/Queue**
   - Daily job to move DEFAULT → GUARANTOR_PHASE & update TI loss.
   - Monthly reset of endorsement capacity.
3. **Security & Compliance**
   - Store all files in GridFS with reference IDs only.
   - Encrypt sensitive fields such as PAN when added.
4. **Testing**
   - Jest/Supertest for each endpoint, especially contract state machine.
5. **Future Scope**
   - PAN KYC micro-service.
   - Smart-contract port if moving to a blockchain later.

This blueprint covers the principal data models and routes; you can now scaffold controllers and business logic on top.