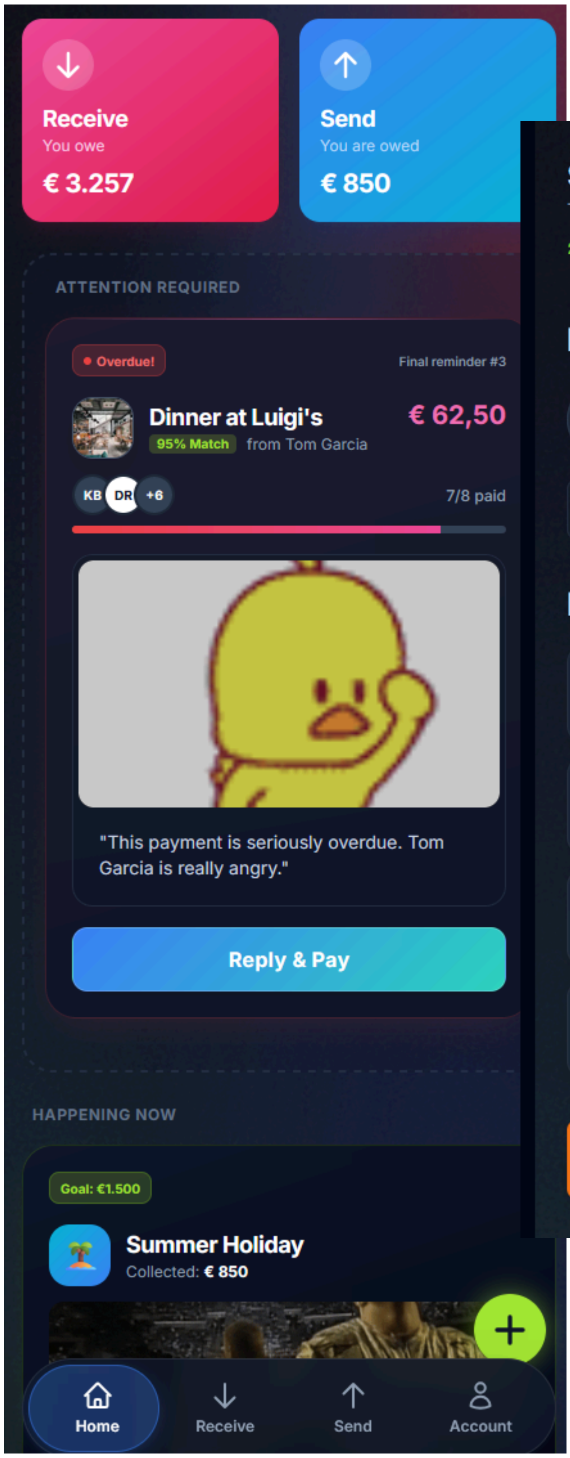
MAXI Product Specification and Technical Guide



Receive
You owe
€ 3.257

Send
You are owed
€ 850

ATTENTION REQUIRED

Overdue!
Final reminder #3

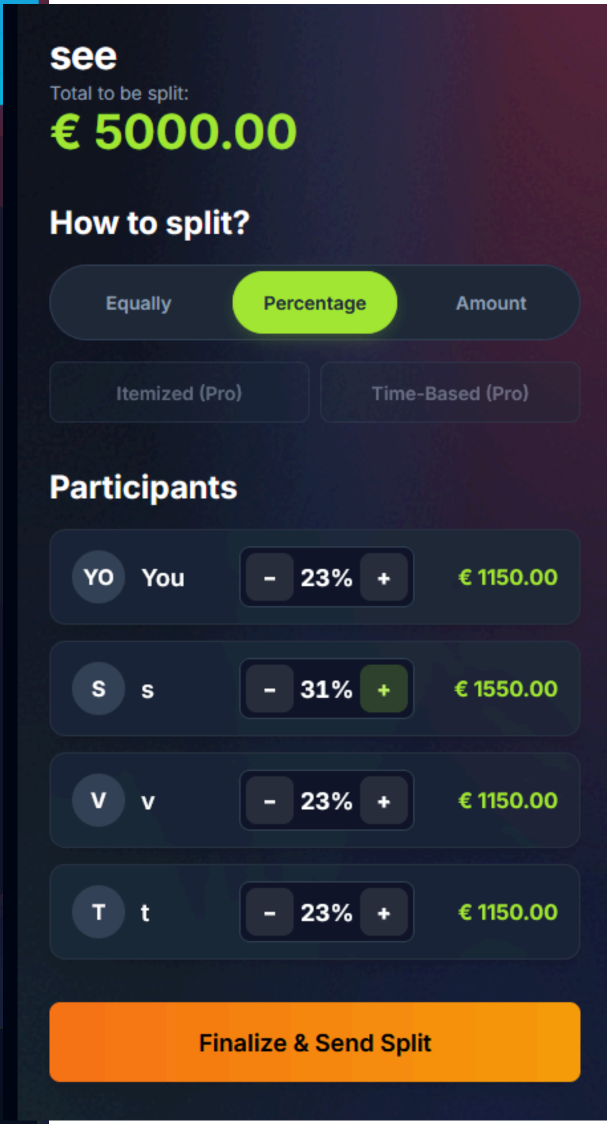
Dinner at Luigi's
€ 62,50
95% Match from Tom Garcia

KB DR +6
7/8 paid

"This payment is seriously overdue. Tom Garcia is really angry."

HAPPENING NOW

Goal: €1.500
Summer Holiday
Collected: € 850



see
Total to be split:
€ 5000.00

How to split?

Participants

YO	You	- 23% +	€ 1150.00
S	s	- 31% +	€ 1550.00
V	v	- 23% +	€ 1150.00
T	t	- 23% +	€ 1150.00

MAXI Product Specification and Technical Guide
1. Product Vision and Core Strategy
    1.1 The Approach: Enticement over Enforcement
    1.2 The "Locomotive" Growth Strategy
    1.3 Single Core, Dual Skin
2. User Identity and Onboarding Logic
    2.1 The Unified Data Model (PostgreSQL)
    2.2 The Identity State Machine
    2.3 The "3-Step Slick" Onboarding
    2.4 Deep Linking and Context Preservation
3. The Core Transaction Engine
    3.1 Universal Payment Link Engine
    3.2 Hotkey Protocol & State Machine
    3.3 Smart Netting & Consolidation
    3.4 AI-Powered Response Engine (Pro Feature)
4. Notification Orchestrator (The "Brain")
    4.1 The Promise Lock
    4.2 Smart Scheduling
    4.3 Event-Driven Architecture
5. MAXI P2P Features ("Social Skin")
    5.1 The Timeline Dashboard
    5.2 Consolidation Splitter (The Event)
        5.2.1 Smart Netting and Consolidation Logic
    5.3 Group Pots ("Bierpot")
    5.4 Advanced Receipt Scanning
6. Notification Orchestrator and Intelligent Scheduling
    6.1 The Promise Lock (Logic of Suppression)
    6.2 Smart Scheduler and Behavioral Optimization
    6.3 Event-Driven Architecture
6. Financial Infrastructure
    6.1 Open Banking (PSD2)
    6.2 Request-to-Pay (SEPA/ISO 20022)
    6.3 Peppol and ViDA compliance
7. Technical Stack
    7.1 Technical stack
    7.2 Data Model Schema (PostgreSQL)
    7.3 API Architecture and Endpoints
8. Security & Data Privacy
    8.1 OWASP MASVS-L2 Security Standard
    8.2 GDPR and Data Privacy
9. Deep Dive: The "Ghost to Owner" State Machine
        State 1: The Ghost (Implicit User)
        State 2: The Guest (Validated User)
        State 3: The Owner (Secured User)
    9.1. Pre-Signup vs. Post-Signup Experience
10. Monetization Strategy and Roadmap
    10.1 Tiers
    10.2 Development Roadmap
11. User Experience Flow Diagram

1. Product Vision and Core Strategy

The MAXI platform is designed to solve a specific problem: the social and financial friction of asking for money. Existing tools act as simple calculators or ledgers. MAXI acts as a behavioral engine. We don't just want a transaction; we want a **Payment Promise**. By turning a cold payment request into a warm, social interaction, we replace the stress of "nagging" with the effectiveness of "enticement."

1.1 The Approach: Enticement over Enforcement

Traditional financial tools rely on enforcement—sending a bill and then escalating demands (dunning) until payment occurs. This damages relationships. MAXI uses **Enticement**. We believe liquidity moves faster when relationships are preserved. We achieve this through the **MaxCredible Score**, a behavioral index that measures communication reliability rather than wealth. A user who clicks "Promise to Pay on Friday" and keeps that promise is valued just as highly as someone who pays instantly. This gives the sender certainty ("I know when the money is coming") without the social cost of chasing.

1.2 The "Locomotive" Growth Strategy

We do not ask users to sign up before they see value. We use the Payment Request or payment request itself as the **"Locomotive"**—the driving force that pulls users into the system. This is **Camouflaged Onboarding**.

- **For Receivers:** They receive a beautiful, interactive request via SMS/WhatsApp. They view it and interact with it (e.g., "Promise to Pay") *before* creating an account.
- **For Senders:** A user can download the app and draft their first Payment Request or split immediately as a "Guest." We only ask them to create an account (save their progress) at the moment they hit "Send," leveraging their investment in the content they just created to ensure conversion.

1.3 Single Core, Dual Skin

To serve two markets efficiently, we use a **Single Core, Dual Skin** architecture.

- **Backend:** One unified engine handling identity, payments, and notifications.
- **Frontend:** Two distinct interfaces.
    - **MAXI Business (SME):** Professional, crisp, focused on cash flow and peace of

mind.
    - **MAXI Social (P2P):** Fun, expressive, focused on shared memories and settling debts.
- **The Bridge:** Because the identity is unified, a freelancer can send a business Payment Request to a client, and that client can open it in the Social skin to split the cost with friends.
    - **Address Book is Shared**. If I have a client "John Doe" in my SME profile, he should appear as a suggestion when I switch to P2P to split a beer, but tagged as "Client" (with a warning prompt: "You are sending a P2P request to a Business Contact").

1.4 Pain point vs solution

We do not build features for the sake of functionality; we build them to solve specific emotional and logistical friction points inherent in social finance.

The Pain Point (The Problem)	**The MAXI Solution (The Feature)**
"The Nagging Factor" Asking friends for money feels like enforcement. Chasing them ("Have you paid yet?") is awkward, conflict-ridden, and creates social tension.	**Automated Reminder Profiles & The "Promise Lock"** We replace manual nagging with "set and forget" profiles (e.g., "Funny," "Polite"). Crucially, the **Promise Lock** stops all reminders the moment a friend clicks "I promise to pay on Friday," preserving the friendship.
"Financial Uncertainty" A "Pending" status is useless. Not knowing *when* money is coming creates anxiety and cripples cash flow planning.	**The Payment Promise (Commitment)** We track the *intent* to pay, not just the payment. A "Promise to Pay on [Date]" is treated as a structured data point, turning uncertainty into a predictable cash flow forecast on the timeline.
"The Sterile Bill" Sending a generic payment link feels cold and transactional, often signaling distance in a relationship.	**The Media Composer & Social Feed** We treat requests as "Shared Memories." By anchoring every bill with a photo/GIF and enabling comments ("Best sushi ever!"), we turn a transaction into a social interaction.

"Group Chaos" Collecting receipts after a trip involves endless texts, lost paper scraps, and confusion over who owes whom.	**The Consolidation Splitter (Time-Boxed)** A "Digital Receipt Bowl" with a countdown timer (e.g., 48 hours). It forces a resolution window, collects all receipts in one place, and uses **Smart Netting** to calculate one final math-perfect settlement.
"I'm Broke Right Now" Friends often don't pay immediately because of liquidity issues, leading to silence and avoidance.	**The "Promise" & "Settle In Natura"** We give them a way to respond *without* paying. They can select a future date (Promise) or offer a non-monetary service ("I'll drive next time") to clear the debt socially.
"The Unfair Split" Splitting a bill equally is unfair when one person had a salad and another had steak and cocktails.	**Itemized Distribution (OCR)** Users can scan a receipt and tap specifically on the items they consumed. The system automatically calculates their exact share, including tax and tip.
"App Fatigue" Users refuse to download a new app just to pay a single one-off debt to a friend.	**The "Locomotive" (Camouflaged Onboarding)** Receivers can view, comment, and even "Promise to Pay" via a secure Web View (Magic Link) without ever creating an account or downloading the app.

2. User Identity and Onboarding Logic

We replace the standard "Sign Up / Log In" flow with a **State-Based Identity Machine**. This allows users to exist in the system with varying levels of validation, supporting the "Locomotive" strategy where users utilize the app before they legally "register."

2.1 The Unified Data Model (PostgreSQL)

The database tracks a single User entity anchored by a mobile phone number (E.164 format). This unified identity allows a single user to traverse multiple contexts (Business and Social) without relinking bank accounts or re-verifying identity.

Table: users	**Description**
id (UUID, PK)	Unique User Identifier.
phone_e164 (VARCHAR)	Unique Mobile Number (Indexed). The primary anchor for identity.
auth_state (ENUM)	IMPLICIT, VALIDATED, SECURED.
max_credible_score (INT)	0-100, behavioral trust index, recalculated nightly.

Table: profiles	**Description**
id (UUID, PK)	Unique Profile Identifier.

user_id (UUID, FK)	Link to User Entity.
type (ENUM)	SME (Business), P2P (Social).
data (JSONB)	Flexible storage (VAT, Brand Colors for SME; Avatar, Display Name for P2P).

2.2 The Identity State Machine (Ghost to Owner)

The Identity State Machine governs the transition between user levels. It ensures that friction (security checks) is introduced only when the user's investment in the platform creates sufficient motivation to overcome it.

State 1: The Ghost (Implicit User)

- **Definition:** A person who has received a Payment Request but has not yet authenticated.
- **Trigger:** Instantiated when a Sender dispatches a request to a phone number not currently in the database.
- **Data Storage:** A "Shadow Record" containing only the hashed phone number and the association with the incoming transaction ID.
- **Capabilities:** Read-Only. They can view the Payment Request via a secure, hash-based "Magic Link" (Web View). They can view the "Hero Media" and "Tone of Voice" content.
- **GDPR Compliance:** Shadow records are hashed and purgable if no conversion occurs within 30 days.

State 2: The Guest (Validated User)

- **Definition:** A user who has proven ownership of their device but has not performed a high-risk action (sending money/requests).
- **Trigger:** The user attempts a write-action (e.g., clicking "Promise to Pay" or commenting). The system triggers an **On-Demand Step-Up**, sending a 6-digit OTP via SMS/WhatsApp.
- **Data Storage:** Local-first (WatermelonDB). Data resides primarily on the device until the account is Secured.

- **Capabilities:** Interactive. Can post comments, select promise dates, and settle bills via external payment methods (iDeal/PISP).
- **Limitation:** Cannot send new Payment Requests.

State 3: The Owner (Secured User)

- **Definition:** A fully onboarded user with a persistent, cloud-synced account.
- **Trigger:** The user creates their own Payment Request (using the "Locomotive" strategy) and taps **"Send."** The system prompts: *"Secure your account to save this request."* This triggers **Biometric Enrollment** (FaceID/TouchID).
- **Data Storage:** Full Cloud Sync (PostgreSQL). Data is preserved across devices.
- **Capabilities:** Full Access. Can create requests, link bank accounts for payouts, and access the SME/P2P switching logic.

2.3 The "3-Step Slick" Onboarding

This flow disguises compliance checks as value-creation steps during the user's first "Send" action.

Step	**User Action**	**System Process (Backend Logic)**
1. Trigger	User drafts a custom request (adds photo, cost) and taps "Send."	System initiates POST /v1/requests/draft. Validates input. Returns 401 Unauthorized with requirements payload: ["identity", "kyb"].
2. Identity	User enters Mobile Number & OTP.	System validates OTP. Upgrades state from **Ghost** to **Guest**. Creates temporary User record.
3. Enrichment	User types Business Name (SME) or Display Name (P2P).	**SME:** Calls Google Places/Corporate Registry API to auto-fill address & VAT. **P2P:** Checks contact graph for existing connections/avatar.
4. Confirmation	User confirms via Biometrics (FaceID).	System upgrades state to **Owner**. Creates Profile. Dispatches the queued Payment Request. Returns 200 OK.

2.4 Deep Linking and Context Preservation

To support Camouflaged Onboarding, the mobile application utilizes a robust **Deferred Deep Linking** architecture. This ensures the user never loses their place.

- **Deferred Logic:** If a Ghost clicks a Payment Request link but does not have the app, they are routed to the App Store. Parameters (request_hash, sender_id) are preserved through installation (via Branch.io or Install Referrer API).
- **Hydration:** Upon first launch, the app initializes, checks for deferred parameters, and immediately hydrates the state to show the specific Payment Request the user was originally sent. They land on the content, not a "Sign Up" screen.

3. The Core Transaction Engine

The Core Engine treats every financial interaction—whether a corporate invoice or a split dinner bill—as a state-managed **Transaction Object**. This module manages the lifecycle of these objects to ensure certainty, enforcing the "Payment Promise" logic and executing seamless settlement.

3.1 Universal Payment Link Engine

To reduce friction, we generate a unique, stateless URL (e.g., maxi.maxcredible.com/pay/xyz123) for every transaction. The engine detects the user agent to serve the appropriate interface:

- **Live State:** The link queries the backend for the current status (Open, Paid, Promised) on load. This prevents double payments.
- **Smart Routing Logic:**
    - **IF User-Agent == Mobile App:** Deep link to the native payment screen (preserves user session).
    - **IF User-Agent == Browser:** Serve a lightweight **Web Payment Client (SPA)**. This view allows "Ghost" users to use Hotkeys (Pay, Promise, Dispute) and pay via Apple Pay/Google Pay without app installation.

3.2 Hotkey Protocol & State Machine

We track "Intent to Pay" as a distinct state. To replace the friction of typing, we use **Hotkeys**—standardized, touch-optimized action buttons that map directly to backend states.

Hotkey Label	**User**	**Backend State Transition**	**System Logic & Side Effects**
PAY NOW	Receiver	VIEWED → PAID	Initiates PISP/Stripe Gateway. On success: updates ledger, stops dunning, sends receipt.
PROMISE TO PAY	Receiver	VIEWED → PROMISED	Opens Date Picker. Sets promise_date. Activates **Promise Lock** (stops reminders until date).
DISPUTE	Receiver	VIEWED → DISPUTED	Suspends all dunning. Opens the "Social Feed" for resolution conversation.

SETTLE IN NATURA	Receiver	VIEWED → PENDING_APPROVAL	**(P2P Only)** User offers non-monetary settlement (e.g., "I'll drive next time"). Sender must approve to clear debt.
MARK AS PAID	Sender	OPEN → PAID_EXTERNAL	**(Sender Override)** Used when paid via cash/external transfer. Stops dunning. Logs "Settled Externally" in timeline.

3.3 Smart Netting & Consolidation Engine

This engine runs whenever a new transaction creates a debt between entities with existing history. It models the financial network as a directed graph to solve the **Minimum Cost Flow** problem.

3.3.1 Graph Simplification Logic

- **Bidirectional Netting:**
    - *Scenario:* User A owes User B €50. User B requests €20 from A.
    - *Action:* Instead of two transactions, the system calculates the net edge: **A owes B €30**.
- **Triangle Simplification (Group Optimization):**
    - *Scenario:* A owes B €10, and B owes C €9.
    - *Action:* The system detects the path A -> B -> C and suggests a direct transfer **A -> C €10**, removing B from the chain to save fees and friction.

3.3.2 The "Penny Perfect" Split Algorithm

When splitting bills, we strictly avoid floating-point errors by calculating in cents and using the **Largest Remainder Method**.

- *Problem:* €9.00 divided by 3 users = 3.3333...
- *Step 1 (Floor):* floor(1000 cents / 3) = 333 cents (€3.33).
- *Step 2 (Remainder):* 1000 - (333 * 3) = 1 cent.
- *Step 3 (Distribute):* The remainder (1 cent) is assigned to the **Bill Owner** (or randomized) to ensure SUM(Shares) == Total exactly.

3.4 AI-Powered Response Engine (Pro Feature)

For P2P contexts where interactions are unstructured, the engine integrates with an LLM to parse natural language into structured Hotkey states.

- **Prompt Engineering Strategy:** Uses "Few-Shot Prompting" to extract Intent and Temporal Variables.
    - *Input:* "I'm broke until my salary hits on the 25th."
    - *Output:* { "intent": "PROMISE", "date": "2025-MM-25", "confidence": 0.98 }
    - *Input:* "I already Tikkied you yesterday."
    - *Output:* { "intent": "DISPUTE", "sub_type": "ALREADY_PAID", "channel": "EXTERNAL" }

4. Notification Orchestrator (The "Brain")

We replace dumb cron jobs with an intelligent **Notification Orchestrator** running on a queue system (BullMQ on Redis).

4.1 The Promise Lock

The most critical functional requirement of the mailer is the **Promise Lock**, which enforces the philosophy of "Respecting the Promise." Before any scheduled reminder is dispatched, the Orchestrator queries the current state of the transaction.

- **Logic:** IF transaction.state == PROMISED AND transaction.promise_date > current_timestamp THEN SUPPRESS_NOTIFICATION.
- **Rationale:** Reminding a user who has explicitly promised to pay on a future date constitutes a social violation and degrades the MaxCredible trust score. It transforms the platform from a helpful tool into a nuisance.
- **Escalation:** IF current_timestamp > transaction.promise_date AND transaction.state!= PAID, the lock acts as a trigger for escalation. The system dequeues the "Friendly" reminder and enqueues an "Urgent/Broken Promise" notification, shifting the Tone of Voice automatically.

4.2 Smart Scheduling

The scheduling logic uses a distributed task queue (e.g., BullMQ on Redis) to manage millions of potential notification events. This choice allows for high-performance job management, including delayed jobs, repeatable jobs, and priority queuing.

- **Social Hours Optimization:** For P2P contexts, the scheduler prioritizes evening and weekend delivery when social engagement is higher, maximizing the open rate for "split bill" requests.
- **Event-Driven Dequeuing:** Unlike static cron jobs that run at fixed intervals, the Redis-based queue allows for dynamic removal. If a webhook from the Open Banking provider confirms a payment at 08:55, the reminder scheduled for 09:00 is instantly identified by its jobId (derived from the transactionId) and removed from the queue. This real-time reactivity prevents the embarrassment of chasing a paid debt.

4.3 Event-Driven Architecture

To ensure scalability and responsiveness, the notification system employs an **Event-Driven Architecture (EDA)**.

- **Event Producers:** The Transaction Engine emits events (e.g., TRANSACTION_CREATED, PAYMENT_PROMISED, PAYMENT_RECEIVED) to an event bus (e.g., Amazon EventBridge or Redis Streams).
- **Event Consumers:** Decoupled microservices consume these events. The Notification Service subscribes to these events to schedule or cancel communications. This decoupling ensures that the core ledger performance is not impacted by the heavy lifting of email/SMS rendering and dispatch.

5. MAXI P2P Features ("Social Skin")

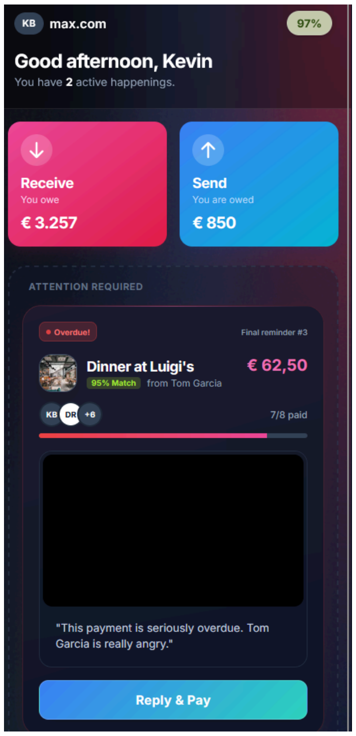
The P2P application ("MAXI Social") is designed for friends, roommates, and social groups. The design philosophy is "Facility for Nonsense," encouraging expressive, non-sterile interactions.

- **Consolidation Splitter:** For group trips. You set a timer (e.g., 48 hours). Everyone dumps their receipts into the app. The Creator acts as a "Bouncer," approving or rejecting receipts. When the timer dings, the app calculates exactly who owes what.
- **Bierpot (Group Pot):** A shared digital wallet. Great for housemates. You can set it to "Auto-Fill" (everyone pays €20/month) or "Open" (throw in cash for pizza whenever).
- **Settle in Natura:** Sometimes you don't want cash. You want a favor. A user can offer "I'll drive next time" to settle a debt. The lender clicks "Accept," and the debt is wiped.

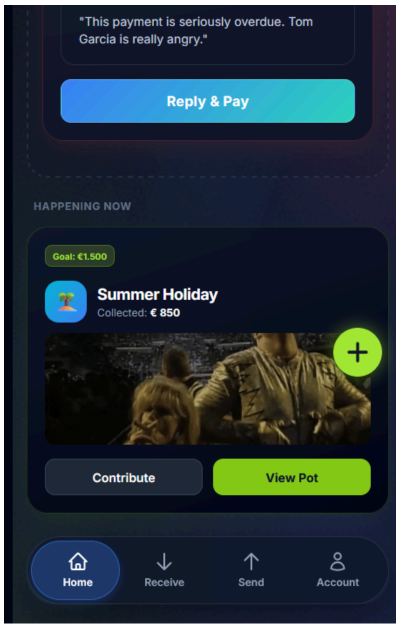
Vibe: "Facility for Nonsense." Fast, fun, expressive.

5.1 The Timeline Dashboard

Unlike the SME ledger, the P2P home screen is a chronological **Timeline** of social-financial events ("You paid Sarah," "Dave requested Pizza money"). Items in the feed are social objects; users can "Like" a payment or comment on a bill. This transforms the app from a utility into a social network for financial interactions.



"This payment is seriously overdue. Tom Garcia is really angry."



"This payment is seriously overdue. Tom Garcia is really angry."

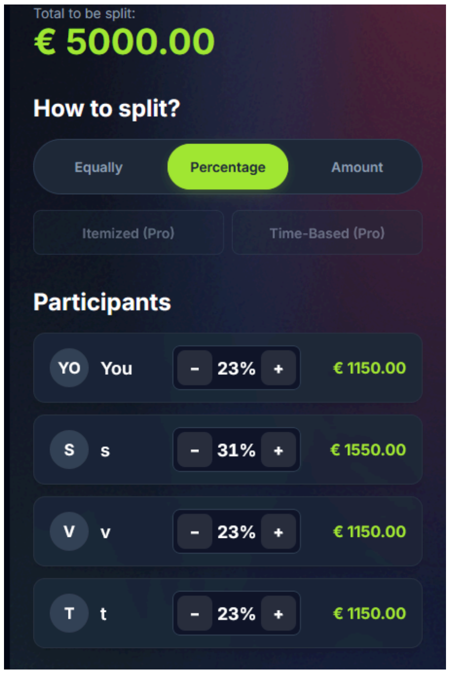
Example: Instead of just "Dinner: €50," the entry should look like a social post: "Sarah created 'Dinner at Luigi's'" → "Mike paid €20" → "You promised to pay Friday.".

5.2 Consolidation Splitter (The Event)

Handles complex events (trips, parties) as a time-boxed session.

- **Timer:** The Creator sets a window (e.g., 48 hours).
- **Receipt Bowl:** Participants upload receipts during this window.
- **Admin Gatekeeper:** The Creator *must approve* every receipt before it enters the final calculation (prevents disputes).
- **Settlement:** When the timer ends, the Smart Netting engine runs, generating the final "who owes who" list.

The splitter requires a smart netting algorithm and a few core features:
-Split equal
-Split by percentage
-Split by custom amount


Total to be split:
€ 5000.00
How to split?

Participants

YO You	23%	€ 1150.00
S s	31%	€ 1550.00
V v	23%	€ 1150.00
T t	23%	€ 1150.00

Smart Splitting Algorithm (Penny Perfect): When splitting an amount (e.g., €100.00) by 3 users, standard division yields €33.333... infinite. **Logic:**

1. Calculate floor share: `floor(100.00 / 3) = 33.33`.
2. Calculate remainder: `100.00 - (33.33 * 3) = 0.01`.
3. Distribute remainder: The €0.01 is assigned to the "Owner" (Creator) or randomized to one participant to ensure `Sum(Shares) == Total`. **Percentage Logic:** Dynamic adjustment ensures that if User A modifies their share to 40%, the remaining 60% is redistributed proportionally among Users B and C, maintaining the locked total.

5.2.1 Smart Netting and Consolidation Logic

The Payment Consolidation Engine uses a **Minimum Cost Flow** algorithm to address the clutter of small debts.

- **Bidirectional Netting:** If A owes B €50, and B owes A €20, the system reduces this to a single edge: A owes B €30.
- **Triangle Simplification:** If A owes B €10, and B owes C €10, the system suggests A pays C €10 directly.
- **Penny-Perfect Math:** When splitting odd numbers (e.g., €9.00 / 3), the system uses the "Largest Remainder Method" to assign the extra €0.01 to the bill creator to ensure `SUM(splits) == Total`.

Alternatively the **Payment Consolidation Engine** can be used for addressing the clutter of multiple small debts, particularly in P2P contexts. It runs a **Smart Netting** algorithm whenever a new transaction creates a cycle or bidirectional edge between entities.

- **Graph Theory Application:** The financial network is modeled as a directed graph where users are nodes and debts are weighted edges. The problem of simplifying debts is a variation of the "Minimum Cost Flow" problem. The algorithm seeks to minimize the total flow (money moved) and the number of edges (transactions) required to satisfy all node balances.
- **Algorithm:** Net Position = Sum(Incoming) - Sum(Outgoing).
  - *Scenario:* User A owes User B €50. User B requests €20 from User A for a new expense.
  - *Naive Approach:* Two transactions (€50 A->B, €20 B->A).
  - *MAXI Approach:* The system detects the cycle. It calculates the net edge: A owes B €30. It proposes a consolidation: "Consolidate debt? New balance: A owes B €30."

- **Group Splitting:** For group expenses, the algorithm simplifies the graph further. If A owes B €10, and B owes C €10, the system can suggest a direct transfer from A to C €10, bypassing B completely. This reduces transaction fees and social friction.

5.3 Group Pots ("Bierpot")

A persistent, shared digital wallet for defined user groups (e.g., "Team Fees," "House Supplies"). For partners, date budgets, swear jars, roommates or teams.

- **Models:** "Open" (anyone contributes anytime), "Scheduled" (monthly rent/fees), "Goal" (birthday gift).
- **Logic:** Tracks a virtual balance. "Money Out" is currently handled by the Admin spending personally and logging the expense against the pot.
- Fun notes:
    - Especially for the swear jars it’s excellent to track swearing and history making it super fun and interactive.
    - For the household it can be super nice to track money out as well (people paying for what exactly)? Toilet paper again?

Goal: To manage the complexity of "Many-to-Many" communication in larger groups (e.g., a 10-person holiday).

Functional Requirements:

1. The "Admin Megaphone" (Pinned Announcements)

- **Requirement:** The Sender (Admin) needs a way to cut through the "nonsense" for critical updates.
- **Feature:** The Admin can "Pin" a message to the top of the Group Feed.
- **Use Cases:**
    - "Deadline for receipts is tomorrow at 5 PM!"
    - "Here is the link to the photo album."

2. Smart "Nudge" Reactions

- **Requirement:** A low-friction way to chase people who haven't paid, without sending a formal "Angry Reminder."
- **Implementation:** In the Group Participant list, next to a user's name (e.g., Mike -

Pending), there is a "Nudge" icon (👋).
- **Action:** Tapping it sends a pre-written, lighthearted notification to the chat: "👋 *Mike, the group is waiting for you!"*

3. Dynamic Group Titling & Avatars

- **Requirement:** The group identity must be flexible. Any participant should be able to update the "Group Cover Photo" (unless locked by Admin) to reflect the current vibe.
- **Logic:** If a user uploads a photo of the group eating dinner, the app suggests: "Set this as group cover?" This keeps the visual identity fresh and relevant to the specific event.

4. Privacy & Mute Controls

- **Requirement:** Users must have control over their notification density.
- **Settings:**
    - **"Mute Chat, Keep Finance":** User receives notifications *only* for new bills or payment requests, but ignores the text/GIF banter.
    - **"Quiet Hours":** Automatically mutes group "nonsense" during work hours (09:00 - 17:00), delivering a daily summary instead.

5.3.2 Group social feed

Goal: To transform the "Audit Log" into a "Conversation." This feed acts as the central timeline for any Bill, Split, or Pot. It merges financial events (System) with human interaction (Chat) to create a single narrative of the expense.

Functional Requirements:

1. The Unified Timeline (Merged Streams)

- **Requirement:** The feed must interleave **System Events** and **User Messages** chronologically. We do not separate "Chat" from "History."
- **Display Logic:**
    - *System Event:* "Kevin paid €20.00" (Styled neutrally/small).
    - *User Message:* "Thanks for paying so fast!" (Styled prominently as a chat bubble).
    - **Rationale:** This ensures that conversation always happens in the *context* of the financial action. A "Thank you" makes no sense if you can't see the "Payment" immediately preceding it.

2. Rich Media "Nonsense" Support

- **Requirement:** To fulfill the "Facility for Nonsense" philosophy, the chat must support high-engagement media types.
- **Supported Media:**
    - **Voice Notes:** A dedicated microphone button for rapid explanations (e.g., explaining a complicated split logic: "I removed the alcohol cost for Mike because he didn't drink").
    - **Direct GIF Search:** Integrated GIPHY support to allow users to react to high bills with humor (e.g., a "fainting" GIF when seeing a total).
    - **Photo Dumps:** Allowing multiple photos to be uploaded in a grid layout within a single chat bubble.

3. Contextual Threading (Item-Level Chat)

- **Requirement:** To prevent group chaos, users must be able to discuss *specific items* without clogging the main feed.
- **Interaction:**
    - User taps on a specific receipt line item (e.g., "Bottle of Vodka - €80").
    - Selects "Discuss Item."
    - **UI:** A sub-thread opens.
    - **Result:** The main feed shows a summary: *"3 comments on 'Bottle of Vodka'"*. This isolates disputes (negative friction) from the main social stream (positive friction).

4. "One-Tap" Social Reactions

- **Requirement:** Users must be able to acknowledge payments without typing.
- **Implementation:**
    - **On Messages:** Standard emoji reactions (Like, Haha, Heart).
    - **On Financial Events:** Specific reaction capability on system events.
        - *Example:* Kevin pays €50. Sarah reacts with the "Fire" emoji directly on the payment log. This validates the payer socially.

5.3.3 **The Virtual Pot (No-Custody Model):**

To avoid EMI licensing requirements in Phase 1, the "Bierpot" functions as a **Virtual Ledger**, not a custodial wallet.

- **Logic:** Users do not "deposit" money. The Pot tracks a "Promise Balance."

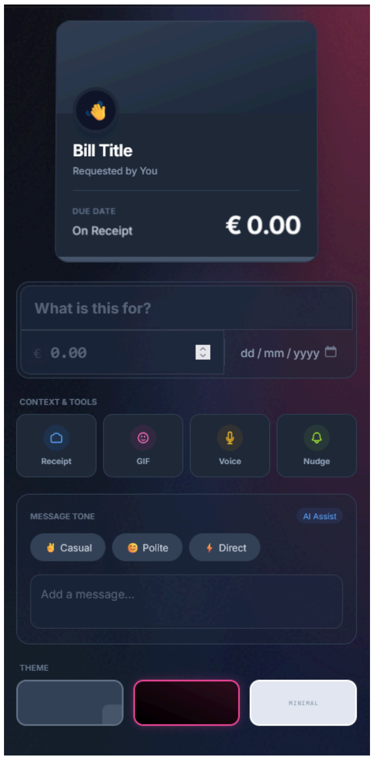
- **Execution:** When the Pot is "Spent" (e.g., Admin pays €100 for pizza), the system triggers a **Batch PISP Request**, instantly pulling the owed amounts from the members' bank accounts and pushing them directly to the Admin's bank account. MAXI never touches the money.

5.4 Advanced Receipt Scanning

Make it easy to start a Maxi experience by starting from the receipt or bill and then adding features on top of that like parsing it on the spot or removing people from the settlement because they contributed something else.

Itemized Distribution: Users can scan a receipt and tap specific items to claim them ("I ate the burger, you had the salad"). The system uses OCR to parse the receipt and automatically splits remaining shared items (tips, tax, appetizers) among the group.

Settle In Natura: A unique social feature allowing a user to propose a non-monetary settlement (e.g., "I'll drive next time" or "I'll clean the kitchen") via the dashboard. If approved by the creditor, the debt is cleared in the system, resolving the social obligation without



5.5 Receive and create awesome Payment Requests, bills, and payment requests

The creation and reception of a request are not administrative tasks; they are the primary social interactions of the platform. We replace the sterile "Form Filling" experience with a "Composition" experience, and the "Bill Paying" experience with a "Social Interaction."

5.5.1 Payment Request creation needs to be state of the art

Goal: To allow a Sender to compose a request that feels like a social media story, not a tax document. The interface must encourage "Enticement over Enforcement".

Functional Requirements:

1. The "Hero" Media Composer (Social Chat Parity)

- **Requirement:** The creation flow must support rich media attachments as the visual anchor of the request. It must support the same media types as a modern chat app (e.g., WhatsApp/Telegram).
- **Supported Media:**
  - **Photos:** Direct camera capture or Camera Roll upload.
  - **GIFs:** Integrated GIPHY/Tenor search API for adding humor.
  - **Voice Memos:** A "Hold to Record" button allowing the Sender to add a verbal context (e.g., "Hey guys, here’s the split for the cabin, totally worth it!").
- **UI Implementation:** The media is not an "attachment" at the bottom; it is the **header/background** of the Payment Request card, ensuring the Receiver sees the memory before the cost.

2. Core Financial Data (The "Hard" Payload)

- **Requirement:** While the look is social, the data structure must be rigorous.
- **Inputs:**
  - **Title (What):** A clear text field for the expense name (e.g., "Dinner at Luigi's").
  - **Amount (How Much):** Large, numeric input.
  - **Due Date (Optional):** A specific date picker. *Logic: Default is "On Receipt" (null).*
  - **Evidence Attachment:** distinct from the "Hero Media," this is a discrete "View Receipt" link for the actual bill/Payment Request image, satisfying the "Admin Gatekeeper" need for proof.

3. The Theming Engine (Visual Templates)

- **Requirement:** The Sender must be able to skin the request to match the vibe of the expense. They should also be able to place each component where they wish.
- **Requirement:** The interface must not be a rigid form. While we offer standard templates for speed, the Sender must have granular control over what is shown to the recipient to ensure every experience is unique to that specific relationship.

- **Templates:**
  - **"Strict/Business":** Minimalist, white background, standard font (for SME-style usage within P2P).
  - **"Party/Fun":** Vibrant gradients, bold typography.
  - **"Apologetic":** Soft colors (for when you feel bad asking).
- **Modular Customization (The Unique Experience):** Beyond templates, the Sender can toggle the visibility of specific blocks:
  - **Visibility Toggles:** The user can choose to **HIDE** the "Due Date" (to reduce pressure), **HIDE** the "Itemized List" (to keep it simple), or **SHOW** only the "Total Amount."
  - **Layout Flexibility:** The user can define the hierarchy—e.g., deciding if the "Hero Media" takes up 80% of the screen (making it a story) or 20% (making it a bill).
- **Technical:** These templates dictate the CSS/Style props of the receiver's web view maxi.maxcredible.com/pay/{hash}.

4. Tone of Voice Manager (Contextual Preamble)

- **Requirement:** A text input area that acts as the "Caption" for the Payment Request.
- **Feature - AI Prompts:** If the user struggles, they can toggle "AI Assist" to generate text based on the selected vibe:
  - *Funny:* "I'm not saying you owe me, but my bank account is crying."
  - *Direct:* "Here is the share for the tickets."
  - *Urgent:* "Please settle this before Friday!"

5. Conditional Reminder Logic (The "Set & Forget" Wizard)

- **Requirement:** Reminder settings must only appear when relevant to reduce UI clutter.
- **Logic:**
  - **IF** Due_Date is **NULL** (On Receipt) --> **HIDE** Reminder Settings. (Assumption: User wants money now; system uses default "nudge" logic).
  - **IF** Due_Date is **SET** --> **SHOW** "Automated Reminders" toggle.

- **Configuration:** If shown, the user can select a "Reminder Profile" (e.g., "Friendly Nudge," "Aggressive Chaser") which dictates the frequency of notifications sent by the Orchestrator.

6. Sender Attribution (Identity)

- **Requirement:** The request must be instantly recognizable to build trust.
- **Display:**
  - **P2P Profile:** Defaults to the user's MaxCredible Avatar and Display Name.
  - **SME Profile:** If the user has an SME profile activated, they can toggle "Send as Business," which swaps the Avatar for the Company Logo and Brand Colors.

5.5.2 Payment Request reception is the core of the platform

Goal: To implement the "Locomotive Strategy". The reception experience must be so engaging that it converts a "Ghost" (guest) into a user without them realizing they are onboarding. The Receiver should feel they are opening a digital gift or message, not a demand for payment.

Functional Requirements:

1. The "Magic Link" Landing (The Reveal)

- **Context:** The user taps a link in WhatsApp/SMS (maxi.com/pay/xyz).
- **Behavior:** The page loads instantly (Web View for Ghosts, Native App for Owners).
- **The "Gift Wrap" Effect:** The first thing visible is **not** the amount owed. It is the **Hero Media** (Section 5.5.1) and the **Sender's Avatar**.
- **Micro-Interaction:** The user sees the photo/GIF and the introductory text ("Tone of Voice") before the financial card slides up or fades in. This establishes social rapport before financial obligation.

2. The "Living" Payment Request (Interactive Feed)

- **Requirement:** The Payment Request is not a static PDF; it is a dynamic room.
- **Social Feed:** Directly below the financial details, the "Comments & Activity" feed is active.
  - *Sender:* "Hope you guys liked the sushi!"
  - *System:* "Bill Created."
  - *Receiver Action:* The Receiver can immediately type a comment ("Loved it!") or react with an emoji without logging in (Implicit User state).

3. The Clarity Module (The Bill)

- **Requirement:** Once the "Social" layer is consumed, the financial request must be unambiguous.
- **Visuals:** High-contrast display of:
  - **Total Amount Owed:** e.g., **€ 62,50**.
  - **Due Date:** e.g., "Due in 8 days" or "Due Immediately."
  - **Status Indicator:** "Pending," "Overdue," or "Paid."

4. The Response Engine (Hotkeys)

- **Requirement:** Providing a response must be easier than ignoring the message. We replace the keyboard with "Hotkeys".
- **Primary Actions (Sticky Footer):**
  - **[PAY & FEEL RELIEF]:** Triggers the Universal Payment Link Engine (Apple Pay, Tikkie, Open Banking).
  - **[PROMISE TO PAY]:** Opens the Date Picker to lock in a specific date. This creates the "Promise Lock" that stops reminders.
  - **[DISPUTE / CHAT]:** "Wait, what is this?" (Opens the Social Feed for discussion).

Types of promises:

Direct Payment: (Highest credibility).

Date Guarantee: "I will pay on [Date]."

Barter Proposal: "Settlement In Natura" (already in your spec, but needs to be classified as a promise type).

Installment: "I will pay 50% now and 50% later" (This appears to be missing from the P2P spec).

Request resettlement: "I don't think this is fair because ...

Types of rejections:

- What is this for?
- I wasn't there
- I disagree because ....

5. The "Camouflaged Onboarding" Hook

- **Logic:** When the Receiver interacts (e.g., sends a Payment Promise or posts a comment), the system triggers the "Just-in-Time" validation.

- **Prompt:** "To secure this promise/comment, verify your number."
- **Result:** The Receiver is now a "Validated Guest" and their MaxCredible Score begins tracking, all initiated by the act of receiving and responding to an awesome Payment Request.

5.6 The Locomotive Transition: Receiver → Sender Conversion

Goal: To convert a Receiver into a Sender not through a "Sign Up" banner, but through the utility and "wonderful experience" of the app itself. The transition must be fluid; the user should feel they are simply "replying" to a financial interaction with one of their own.

Core Philosophy:

- **One User, Two Modes:** A user is always both a Sender and a Receiver. We do not use "Mode Switching" or "Flip Buttons." Both capabilities exist on the same screen.
- **Camouflaged Onboarding:** We do not ask for account creation until the user has *already* done the work of creating a request. We "Save to Send" rather than "Sign Up to Use".

5.6.1 Trigger 1: The "Post-Interaction" Hook (The Dopamine Moment)

- **Context:** This trigger occurs immediately after the Receiver has successfully interacted with an incoming request (e.g., sent a "Payment Promise" or "Paid" via Tikkie).
- **State:** The user feels relief ("Goodbye pending, hello peace of mind").
- **UI Implementation:**
    - Instead of a generic "Success" screen, the system presents a contextual CTA based on the type of bill they just interacted with.
    - *If they just split a dinner bill:* "That was easy. **Split your own receipt?**"
    - *If they just contributed to a Pot:* "**Start a Pot for your group?**"
- **Action:** Tapping this CTA immediately opens the **Composer** (Section 5.5.1) with a fresh, blank template. The user is now a Sender, effectively "drafting" before "registering."

5.6.2 Trigger 2: The "Unified Timeline" (The Persistent Hook)

- **Context:** The Receiver returns to the dashboard to view the status of their debts.
- **UI Implementation:**
    - **The Combined View:** The dashboard is a single chronological timeline showing both "Incoming" and "Outgoing" requests intermingled. This visually reinforces

    that "Sending" is just a normal part of the feed.
    - **The Persistent Plus (+):** A prominent, floating action button is always visible. It is not labeled "Create Account"; it is labeled "**New Request**" or simply **"+"**.
- **Logic:**
    - Tapping **+** allows a "Guest" (unregistered user) to fully utilize the **Media Composer** and **Theming Engine** (Section 5.5.1).
    - They can upload photos, add costs, and write a "Tone of Voice" caption *without* an account.

5.6.3 Trigger 3: The "Catalyst Splitter" (The Viral Hook)

- **Context:** A user receives an **SME Invoice** (e.g., for a €1,200 team dinner) that needs to be shared with others.
- **UI Implementation:**
    1. On the "SME Invoice Detail" screen, a prominent secondary button appears: **[Split this Bill]**.
- **The Flow:**
    1. Receiver taps **[Split this Bill]**.
    2. System instantly transitions them to the **P2P Composer**.
    3. **Auto-Population:** The system pre-fills the "Amount" (€1,200) and "Title" (from the invoice) into the new P2P request.
    4. **Conversion:** The Receiver is now the "Admin" of a new P2P split. They add friends and hit send.

5.6.4 The "Trap": Saving Progress to Send

- **The Mechanism:** This is the critical "Gatekeeper" moment that replaces the registration form.
- **Scenario:** A "Guest" user has just spent 2 minutes creating a beautiful request (Trigger 1, 2, or 3). They have added a funny GIF, typed a caption, and entered the amount. They hit **[SEND]**.
- **The System Response:**
    - The system does *not* send the request immediately.
    - **Prompt:** "Secure your request to track payments?" (or "Save this split to your secure history?").
    - **Action:** The user confirms.
    - **Biometric Lock:** The system triggers the native FaceID/Fingerprint prompt.
- **Result:** The account is created in the background using the validated phone number (from the previous "Promise" step) + Biometrics. The request is sent. The user is now an **Owner**.

6. Financial Infrastructure

To deliver on the promise of "Certainty," MAXI must tightly integrate with the European banking infrastructure, leveraging PSD2 APIs and ISO 20022 standards to close the loop between Payment Request and cash.

6.1 Open Banking (PSD2)

- **Payment Initiation Services (PISP):** For the "Pay Now" hotkey, the system utilizes Payment Initiation Service Provider endpoints. When the receiver selects their bank, MAXI initiates a payment order via the PISP API. This redirects the user to their native banking app (app-to-app deep linking) to authenticate the transfer.
    - **Advantage:** This method "pushes" funds directly from the Payer to the Payee (SME) without the intermediaries and high fees associated with credit cards. It provides instant confirmation of initiation, allowing MAXI to update the Payment Request status to `PROCESSING` immediately.
- **Account Information Services (AISP) and Automated Reconciliation:** For the "SME Pro" tier, the system offers Automated Reconciliation. The system polls the SME's connected bank account for new transaction data (`GET /transactions`).
    - **Multi-Dimensional Matching Algorithm:** To match incoming payments to open Payment Requests, the system employs a sophisticated matching engine:
        1. **Exact Match:** Transaction Amount == Payment Request Amount AND Payment Reference contains Payment Request Number. (Confidence: 100%).
        2. **Fuzzy Match:** Transaction Amount == Payment Request Amount AND Sender Name is phonetically similar to Client Name (using Levenshtein Distance or Soundex algorithms).
    - **Logic:** High-confidence matches automatically transition the Payment Request to `PAID`. Lower-confidence matches flag a "Suggested Match" for the user to confirm manually.

6.2 Request-to-Pay (SEPA/ISO 20022)

To future-proof the application for the 2025 landscape, the architecture includes support for the SEPA Request-to-Pay (SRTP) scheme using **ISO 20022** messaging standards. This moves beyond simple payment links to integrated banking requests.

- **pain.013 (CreditorPaymentActivationRequest):** This message is used to send the structured payment request directly to the payer's bank interface. Key fields include the EndToEndIdentification (to link the request to the future payment) and structured remittance information.
- **pain.014 (CreditorPaymentActivationRequestStatusReport):** This message is used to receive the status (Accepted/Rejected) of the request from the payer's bank.
- **Integration Value:** This allows the SME to send a request that appears directly in the client's banking app, not just their email. When the payer accepts the request in their banking app, the payment is irrevocably initiated, representing the "gold standard" for reducing payment friction in the EU.

6.3 Peppol and ViDA compliance

With the "VAT in the Digital Age" (ViDA) regulations approaching, MAXI ensures B2B compliance through **UBL 2.1** syntax.

- **UBL Mapping:** The backend Payment Request generation engine outputs data in the Universal Business Language (UBL 2.1) XML format. Even if the user interacts with a simple UI, the underlying data structure maps to specific XML tags (e.g., cac:AccountingSupplierParty, cac:TaxTotal, cbc:IssueDate) required for **Peppol BIS Billing 3.0** compliance.
- **E-Invoicing Readiness:** This architecture ensures that MAXI can facilitate the mandatory B2B digital reporting and e-invoicing requirements set to take effect across the EU by 2027/2027. The system is effectively a pre-configured e-invoicing node for SMEs.

7. Technical Stack

The system requires a robust, scalable architecture to handle the unified identity and high-frequency event processing.

7.1 Technical stack

- **Frontend: React Native** is selected for a high-performance, cross-platform experience (iOS/Android) with a shared codebase. Client-side PDF generation (react-native-pdf) is crucial for offline Payment Request creation.
- **Offline Database: WatermelonDB** is chosen over Realm for its lazy loading capabilities and seamless synchronization protocol.
    - **Sync Protocol:** WatermelonDB uses a pull/push protocol. pullChanges fetches created/updated/deleted records from the server since the lastPulledAt timestamp. pushChanges sends local modifications to the server. This is essential for the "Guest Creator" mode, allowing users to draft Payment Requests offline and sync when they convert to a secure account.
    - While the *Sender* works offline, the `POST /draft` sync **must** happen before the Magic Link works for the Receiver. You cannot send a functional link without the server knowing the amount/context.
- **Backend: Node.js** microservices architecture.
- **Queue Management: BullMQ** on Redis is the backbone of the Notification Orchestrator. It is essential for managing the "Promise Lock" logic (removing delayed jobs) and handling high-throughput event streams.
- **Database: PostgreSQL** is the primary data store, chosen for its relational integrity and robust JSONB support, which is vital for the flexible profile data model.

7.2 Data Model Schema (PostgreSQL)

The schema is designed to support the Unified Identity and complex Transaction logic.

Table: transactions (Payment Requests/Splits)

Column	Type	Description
id	UUID (PK)	Unique Transaction Identifier.
sender_profile_id	UUID (FK)	Link to Sender Profile.
receiver_phone	VARCHAR	Captured immediately for Implicit User creation.
amount_cents	BIGINT	Stored in cents to avoid floating-point errors.
status	ENUM	SENT, VIEWED, PROMISED, PAID, DISPUTED.
promise_date	TIMESTAMP	The **"Promise Lock"** trigger (Nullable).
magic_hash	VARCHAR	Unique hash for the public Web View URL.
origin_app	ENUM	sme, p2p (Dictates rendering style in the feed).
audit_log	JSONB	Array of events for the Recipient Status Dashboard.

7.3 API Architecture and Endpoints

The API must be RESTful, secured by JWT, and support the Unified Identity model.

Identity & Onboarding:

- POST /v1/auth/magic-link: Payload { phone }. Checks existence, creates IMPLICIT user if null, sends SMS.
- POST /v1/auth/validate: Payload { otp, session_token }. Upgrades session to VALIDATED.
- POST /v1/user/secure: Payload { biometric_hash }. Upgrades to SECURED.

Transaction Management:

- POST /v1/Payment Requests/draft: Creates initial record.
- GET /v1/Payment Request/{magic_hash}: Public endpoint for Receiver View (requires OTP for full data).
- POST /v1/transaction/{id}/react: Payload { type: 'PROMISE', value: '2025-10-20' }. Triggers State Machine transition and updates Promise Lock.

Financial:

- POST /v1/bank/link: Initiates Open Banking flow (Plaid/GoCardless).
- GET /v1/bank/reconcile: Triggers on-demand reconciliation scan.

8. Security & Data Privacy

8.1 OWASP MASVS-L2 Security Standard

Given the financial nature of the application, the security architecture adheres to **OWASP MASVS Level 2 (Defense-in-Depth)**. This level is appropriate for apps handling highly sensitive data.

- **Root Detection:** The app integrates Runtime Application Self-Protection (RASP) to detect and refuse execution on rooted or jailbroken devices.
- **Certificate Pinning:** Mandatory SSL pinning is implemented to prevent Man-in-the-Middle (MitM) attacks on the API traffic.
- **Secure Storage:** Sensitive tokens (Refresh Tokens, Biometric keys) are stored exclusively in the device's secure enclave (iOS Keychain / Android Keystore), never in local storage or AsyncStorage.
- **Biometric Authentication:** Implementation follows `MASTG-TEST-0018` guidelines, ensuring that biometric unlock keys are cryptographically bound to the device hardware and invalidated if the biometric enrollment changes.

8.2 GDPR and Data Privacy

- **Right to be Forgotten:** The API includes a `POST /v1/user/anonymize` endpoint. This anonymizes PII (Personally Identifiable Information) in the `users` and `profiles` tables.
- **Data Retention Strategy:** While the user profile is anonymized, the integrity of the `transactions` table (Financial Ledger) must be preserved for tax purposes (typically 7-10 years). This is achieved by crypto-shredding the link between the user ID and the personal data, effectively rendering the transaction logs anonymous but valid for auditing. This aligns with GDPR Article 17 exceptions for legal obligations.
- **Privacy Firewall:** The system implements a strict logic gate to prevent social data leakage. A user's P2P "Social Score" is never visible to an SME sender unless the receiver explicitly opts-in.
- **The "Shadow Profile" Constraint:** We store phone numbers of non-users (Ghosts) when a user sends them a request. To comply with GDPR, these "Shadow Records" must be:
    1. Hashed (non-reversible) OR
    2. Purged after 30 days if the Ghost does not convert to a Guest.
    3. Not searchable by other users until the Ghost validates their number.

9. Deep Dive: The "Ghost to Owner" State Machine

We don't have a binary "Signed Up / Not Signed Up" model. We track users through three distinct stages of reality. This allows us to let people use the app *before* they fully commit.

State 1: The Ghost (Implicit User)

- **Definition:** A person who has received a payment request but hasn't touched the app yet.
- **Data Held:** Just a phone number (stored as a "Shadow Record").
- **Capabilities:** They can view an Payment Request via a secure "Magic Link" we text them. They can read the chat but cannot write.
- **Tech Check:** When they click the link, we generate a temporary read-only token.

State 2: The Guest (Validated User)

- **Definition:** A Ghost who wants to interact. Maybe they clicked "Promise to Pay" or "Dispute."
- **The Trigger:** To do this, they must prove they own the phone number. We send a 6-digit SMS code.
- **Capabilities:** Once they enter the code, they upgrade to **Guest**. They can now type comments, select promise dates, and pay bills.
- **Limitation:** They cannot *send* new requests yet. They are only responders.

State 3: The Owner (Secured User)

- **Definition:** A Guest who wants to start sending their own requests (SME or P2P).
- **The Trigger:** When they draft a bill and hit "Send," we ask them to "Secure Account." This triggers Biometric enrollment (FaceID/TouchID).
- **Capabilities:** Full power. They can create Payment Requests, link bank accounts, and withdraw money. They are now a permanent node in our network.

9.1. Pre-Signup vs. Post-Signup Experience

The magic of MAXI is **"The Locomotive Strategy."** We use the Payment Request itself to pull people in, rather than forcing a registration form on them.

Feature	**Pre-Signup (Guest Sender)**	**Post-Signup (Owner)**
Data Storage	Local Only (WatermelonDB). If they delete the app, data is lost.	Cloud Synced (PostgreSQL). Data is safe forever.
Payment Request Creation	Can draft fully custom Payment Requests with photos and logos.	Can send unlimited Payment Requests and track history.
The "Send" Action	**The Trap:** When they hit "Send," we prompt: "Save progress to send?" This creates the account.	**The Flow:** Sending is instant and background-synced.
Receiving Money	Can see money is "Waiting," but cannot touch it.	Can link a bank account and withdraw funds (Payouts).

This means a user can download the app, spend 10 minutes making a beautiful Payment Request, and feel "invested" in it. When we finally ask for their phone number to send it, they do it gladly because they don't want to lose their work.

10. Monetization Strategy and Roadmap

The monetization model is a **Freemium Funnel** designed to upsell users from the viral P2P loop into the paid SME ecosystem.

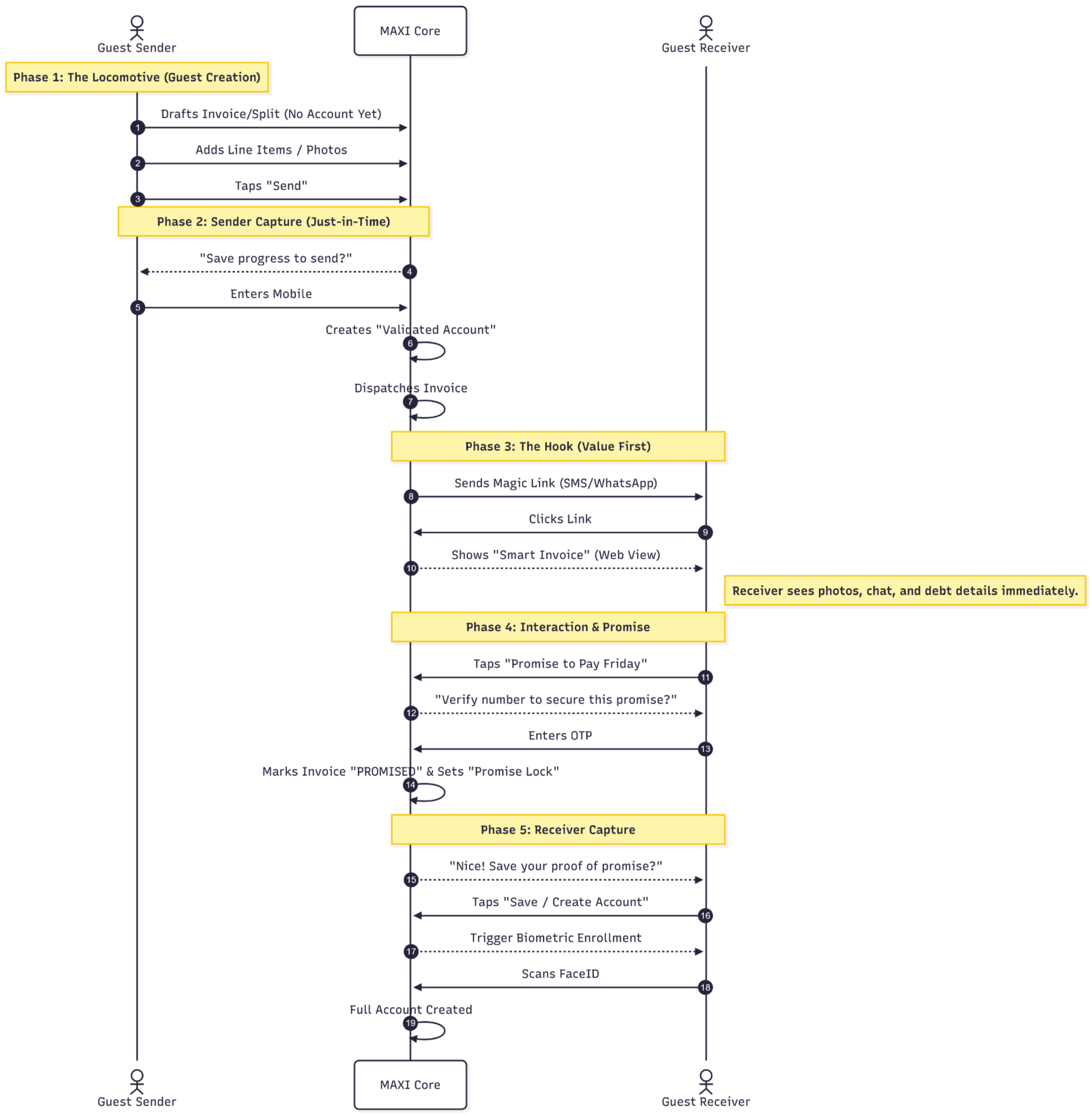
10.1 Tiers

- **P2P Free:** Ad-supported, standard splitting features.
- **P2P Pro:** Ad-free, AI-Powered Hotkeys, Itemized Distribution (OCR).
- **SME Free (Starter):** Limit of 3-7 Payment Requests per month. Mandatory "Sent with MAXI" footer (branding). Manual payment links only.
- **SME Essential (Pro):** Unlimited Payment Requests, White Label (No footer), Integrated Payments (Stripe/PISP) for one-click payment, "Strict" Reminder Profiles.
- **SME Enterprise:** API Access for accounting software sync (Quickbooks, Xero, AFAS), Automated Reconciliation Engine (AISP matching), and "Factoring" integration ("Get Paid Now" feature).

10.2 Development Roadmap

- **Phase 1: The Core Viral Loop (MVP):** Validate the "Locomotive" and Unified Identity. Deliverables: Implicit User creation, Magic Link Auth, Basic Payment Request Wizard, Mobile Web Receiver View, Hotkey State Machine, Deep Linking infrastructure.
- **Phase 2: The Certainty Engine:** Reduce sender anxiety. Deliverables: Notification Orchestrator (BullMQ), Promise Lock logic, MaxCredible Scoring Algorithm, Pulse Dashboard (Liquidity Graph).
    - Once a user authenticates, we must ingest their phone book to build the "Social Graph" (who knows whom). This is critical for the "Consolidation Splitter" to suggest friends automatically.
- **Phase 3: Financial Integration:** Close the financial loop. Deliverables: Open Banking (PSD2 PISP/AISP) integration, Automated Reconciliation (Fuzzy Matching), Stripe Connect implementation.
- **Phase 4: The "Fun & Power" Ecosystem:** Market differentiation and advanced compliance. Deliverables: AI-powered Hotkeys (NLP), Peppol/ViDA Compliance (XML Export), Catalyst Splitter (SME to P2P conversion feature).

11. User Experience Flow Diagram


Guest Sender
MAXI Core
Guest Receiver
Phase 1: The Locomotive (Guest Creation)
1 Drafts Invoice/Split (No Account Yet)
2 Adds Line Items / Photos
3 Taps "Send"
Phase 2: Sender Capture (Just-in-Time)
4 "Save progress to send?"
5 Enters Mobile
6 Creates "Validated Account"
7 Dispatches Invoice
Phase 3: The Hook (Value First)
8 Sends Magic Link (SMS/WhatsApp)
9 Clicks Link
10 Shows "Smart Invoice" (Web View)
Receiver sees photos, chat, and debt details immediately.
Phase 4: Interaction & Promise
11 Taps "Promise to Pay Friday"
12 "Verify number to secure this promise?"
13 Enters OTP
14 Marks Invoice "PROMISED" & Sets "Promise Lock"
Phase 5: Receiver Capture
15 "Nice! Save your proof of promise?"
16 Taps "Save / Create Account"
17 Trigger Biometric Enrollment
18 Scans FaceID
19 Full Account Created
Guest Sender
MAXI Core
Guest Receiver