Section A — Use Case Overview

OysterHR serves a global network of freelancers who expect to get paid quickly, reliably, and in stable currency. But today's payout infrastructure slows them down: bank transfers settle in days, cross-border payments are expensive and fragmented, and freelancers frequently face delays accessing their earnings.

On OysterHR's side, supporting instant, programmable, global payouts introduces significant complexity:

- **Settlement on banking rails is slow.** ACH, card networks, and international payment systems batch transactions and introduce multi-day delays.
- **Money is difficult to program at the banking layer.** Automated payouts, bill splitting, tax withholding, and conditional transfers all require heavy custom logic because banks can't express these behaviors natively.
- **Global payouts require separate integrations per region.** ACH, SEPA, Pix, UPI, and other local networks don't interoperate.
- **Managing wallets or private keys in-house is risky** and requires deep crypto infrastructure experience.
- **Freelancers increasingly expect stablecoins like USDC**, but supporting crypto payout flows adds even more operational steps.
- **Idle funds generate no value**, even though many fintechs (Coinbase, Tether, etc.) are already earning meaningful returns on stablecoin-based balances.

Grid solves these challenges by providing a **programmable smart-account infrastructure** purpose-built for embedded finance. OysterHR can give every freelancer a dedicated on-chain smart account without touching keys, blockchain engineering, or new security models. From there:

1. Instant global payouts

OysterHR can pay freelancers in USDC in seconds, rather than days. Funds are immediately spendable.

2. Fully programmable money

Smart accounts support automatic tax withholding, split payments, recurring payouts, conditional transfers, and more — without OysterHR baking custom logic into multiple banking systems.

3. No blockchain complexity

Grid abstracts MPC key management, Solana transaction signing, account creation, fee handling, and safety controls behind simple APIs and SDKs.

4. Ability to unlock yield

Balances in smart accounts can earn yield through stablecoins or treasury-backed assets, providing new revenue opportunities and greater value for freelancers.

5. A unified settlement layer

Instead of integrating multiple local payment systems globally, OysterHR can use USDC as a common settlement layer — reducing engineering overhead and creating a more consistent freelancer experience.

In short, OysterHR can deliver an instant, global, modern payout experience — without building new financial infrastructure. Grid provides the backbone for a smart-account–based banking experience designed specifically for freelancers.

Section B — Architecture & Flow

This section explains how Grid integrates into OysterHR's existing stack to deliver instant USDC payouts, smart account management, and programmable money flows.

---

1. Account Provisioning

OysterHR creates a smart account for each freelancer using a simple API call:

- Grid provisions a **Solana smart account** on-chain.
- Grid manages the underlying **MPC signer infrastructure**, so OysterHR never handles private keys.

Freelancers don't need seed phrases, wallets, or onboarding flows — the smart account is fully controlled by OysterHR's application through Grid's permissions and access controls.

---

2. Funding Flow: From Payroll to USDC

A typical payroll flow looks like:

1. OysterHR processes a payroll run internally.
2. OysterHR sends USDC (or initiates a conversion flow, depending on upstream rails).
3. The USDC is transferred into each freelancer's Grid smart account using the SDK.
4. The transaction is signed by Grid's MPC signer and executed on Solana.

Funds appear in the freelancer's account in seconds and are immediately usable.

---

3. Spend & Payment Actions

Once USDC is in the freelancer’s account, OysterHR can expose spend actions through the product. Grid supports:

- **Peer-to-peer transfers** (freelancer → other accounts or vendors)
- **Payouts to on-chain addresses**
- **Programmable workflows**, such as:
    - automatic savings
    - bill splitting
    - tax withholding
    - conditional or recurring transfers
- **Interactions with on-chain programs** (if OysterHR wants advanced features like yield or swapping)

All of these actions are executed securely through Grid’s signer, with OysterHR controlling permissions at the application layer.

---

4. Key Management & Security

Grid handles all cryptographic responsibilities:

- Smart accounts are controlled by Grid’s **distributed MPC signer**.
- No private keys exist in a single place.
- OysterHR controls access through Grid’s API-scoped permissions.
- Signatures are authorized only for allowed operations.

This removes an entire category of engineering risk for OysterHR.

---

5. Compliance Considerations

Grid operates as a **non-custodial infrastructure provider**, meaning:

- OysterHR retains full control of user balances and flows.
- Grid never has unilateral access to funds.
- Smart accounts remain on-chain and transparent.
- OysterHR implements KYC/KYB policies appropriate for its jurisdiction.

This setup helps OysterHR stay aligned with regulatory expectations around self-custody and user-controlled funds.

---

6. Scalability & Reliability

Grid is built on top of Solana, enabling:

- High throughput (tens of thousands of transactions per second)
- Extremely low fees
- Fast confirmation times
- Globally consistent settlement

The underlying smart-account framework and MPC signer network scale horizontally, so provisioning tens or hundreds of thousands of freelancer accounts remains lightweight and predictable.

Section C: SDK and API Flow Walkthrough (One-Pager)

This section highlights the core methods and integration points OysterHR would use when interacting with Grid. These examples focus on the essentials needed to provision Smart Accounts, manage spending controls, and automate payroll operations. **Even with just these four methods, Grid provides a clean, predictable developer experience for managing accounts, spending controls, balance monitoring, and automated payroll operations inside the OysterHR product.**

---

1. Creating Smart Accounts (SDK)

OysterHR provisions a Smart Account for each freelancer during onboarding. The SDK abstracts away the underlying Solana account creation and key management.

```
import { GridClient } from "@sqds/grid";


const grid = new GridClient({ apiKey: GRID_API_KEY });


const account = await grid.createAccount({
  email: freelancerEmail
});
```

Purpose:

- Creates a secure, programmable Smart Account
- Associates metadata for internal mappings
- Defines the account where USDC payroll funds will be sent

---

2. Checking Available Balance (SDK)

OysterHR retrieves account balances to display in dashboards or validate that funds are available prior to initiating any payout.

```
const balances = await grid.getAccountBalances(accountAddress);
```

Purpose:

- Retrieve a freelancer's available USDC balance
- Validate pre-conditions before executing payment actions
- Power product interfaces with real-time account information

---

3. Creating Spending Limits (SDK)

OysterHR sets spending controls to enforce budget limits and enable autonomous spending within defined boundaries.

```
const spendingLimit = await grid.createSpendingLimit(accountAddress,
{

  amount: 100000, // $100 USDC monthly limit

  mint: "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v", // USDC

  period: "monthly",

  spending_limit_signers: [freelancerAddress]

});
```

Purpose:

- Set spending controls for freelancer accounts
- Enforce compliance and budget limits
- Enable autonomous spending within defined boundaries

4. Creating Standing Orders (SDK)

OysterHR automates recurring payroll payments, eliminating manual processing and ensuring consistent payment timing.

```
const standingOrder = await grid.createStandingOrder(accountAddress,
{

  amount: "500000", // $500 monthly salary

  grid_user_id: freelancerGridId,

  source: { account: companyAccount, currency: "usdc" },

  destination: { address: freelancerAccount, currency: "usdc" },

  frequency: "monthly",

  start_date: "2024-01-01T00:00:00Z"

});
```

Purpose:

- Automate recurring payroll payments
- Eliminate manual payment processing
- Ensure consistent payment timing

Section D: Next Steps and Future Capabilities

As OysterHR continues integrating Grid, the platform’s planned enhancements could enable more sophisticated financial workflows and richer experiences for freelancers. These expansions would allow OysterHR to gradually unlock additional functionality as Grid evolves.

---

1. Automated Workflows

- Planned support for investment platforms and automated portfolio management.
- Use case for OysterHR: automatically allocate a portion of freelancer payouts to savings or investments.

2. Self-Custodial Cards

- Physical or virtual cards connected to freelancers Smart Accounts.
- Use case for OysterHR: allow instant spending of payroll funds, improving the freelancer experience and reducing delays between payment receipt and usage.

3. Expanded Banking and Payment Rails

- Integration with additional KMS providers and banking networks.
- Use case for OysterHR: support more currencies, cross-border payments, or alternative settlement options to reach global freelancers efficiently.

4. Programmatic, Exportable Statements

- Generate detailed, structured reports for finance teams.
- Use case for OysterHR: simplify bookkeeping, reconciliation, and regulatory reporting as operations scale.