ChatGPT

ChefsPlan: A Decentralized Gig-Work Platform for Chefs and Restaurants

System Overview and Motivation

ChefsPlan is a gig-work platform connecting freelance chefs with restaurants to fill temporary shifts. The platform's mission is to provide a **trustless, decentralized marketplace** for hospitality work, much like an "Airbnb for chefs," while addressing key limitations of traditional staffing platforms. Conventional gig platforms often require users to trust a central intermediary with payments, reputation scores, and sensitive personal data. They also may impose high fees and lack privacy safeguards. ChefsPlan's decentralized architecture aims to solve these issues by using blockchain smart contracts for **escrowed payments** and agreements, **IPFS** for distributed data storage, and **zero-knowledge proofs** for privacy-preserving credential verification. By leveraging zkSync (a Layer-2 rollup on Ethereum) and off-chain automation agents, ChefsPlan ensures that chefs and restaurants can do business with *minimized trust assumptions* and improved security. Both parties gain confidence that funds will be handled fairly and that personal data remains private, while the system remains scalable and user-friendly.

Architecture Overview

Figure: High-level architecture of ChefsPlan. Chefs and restaurants interact via a dApp with smart contracts on zkSync (handling shifts, escrows, and reputation). Job details and documents are stored on IPFS (referenced by content hashes on-chain). Off-chain autonomous agents (Matching, Compliance, Payout, and AI Copilot) monitor the blockchain and external data to orchestrate matching, credential checks, and payouts.

At a high level, ChefsPlan's architecture is composed of **on-chain smart contracts** on the zkSync Layer-2 network, **off-chain decentralized storage** (IPFS) for handling larger data, and **off-chain autonomous agents** that assist with tasks like matching and compliance. End-users (chefs and restaurant managers) interact with the platform through a web or mobile dApp that communicates with the smart contracts on zkSync. When a restaurant posts a new shift (job) or a chef applies for a gig, these interactions are translated into blockchain transactions or off-chain records as appropriate. The **zkSync L2** provides fast, low-cost transactions secured by Ethereum's consensus via zk-Rollup proofs, which enables ChefsPlan to handle many micro-transactions (like escrow deposits, confirmations, and reputation updates) economically. Data that would be costly or impractical to put on-chain (such as detailed job descriptions or legal documents) is kept off-chain in IPFS, with only content hashes stored in smart contracts [1]. Meanwhile, a set of autonomous agents runs off-chain to monitor events and external conditions: for example, a **Matching Agent** watches for new shift postings and suggests suitable chefs; a **Compliance Agent** verifies that participants meet requirements (using off-chain data and zero-knowledge proofs); and a **Payout Agent** triggers automated payouts from escrow when conditions are fulfilled. An AI-driven **Copilot agent** oversees system health and can assist in orchestrating these processes (for example, by coordinating agent tasks or providing analytics insights). This hybrid architecture ensures that core transactions and records are secured on-chain, while heavy data and complex logic can be handled off-chain to maintain scalability and flexibility.

On-Chain Architecture on zkSync

ChefsPlan's on-chain component consists of a suite of **smart contracts on the zkSync Era** network that implement the core marketplace logic: posting shifts, forming agreements between chefs and restaurants, holding payments in escrow, and updating reputation. The contracts follow a modular design, separating concerns for easier maintenance and potential upgrades. Key on-chain modules include:

- **Shift Contract (Agreements Registry):** This contract manages the lifecycle of gig postings and agreements. When a restaurant posts a shift, a new entry (or lightweight contract instance) is created representing the job offer. The posting includes an IPFS content hash pointing to the job's details (role, time, location, pay rate, etc.) rather than storing large text on-chain [1]. Chefs can then apply or express interest in the shift (e.g., by calling an `applyToShift(shiftId)` function or signing an off-chain message that the restaurant can acknowledge on-chain). Once a restaurant selects a chef for the shift, the contract transitions that shift entry into an "accepted" state, effectively forming a binding agreement. At that point, the restaurant is required to deposit the agreed payment into the escrow contract (atomically or in the same transaction flow). Each shift entry in the contract maintains fields such as the employer (restaurant address), the chosen chef's address, the payment amount, a reference to the escrow, timestamps (start/end of shift), and a status enum (Open, Accepted, Completed, Cancelled, Disputed, etc.). The shift contract emits events for critical actions (posted, applied, accepted, completed) which off-chain agents and UIs can monitor.

- **Escrow and Payout Contract:** This contract securely holds funds for a gig and releases them upon successful completion (or refunds if the gig is cancelled). When a shift is accepted, the restaurant's payment (in a stablecoin or zkSync's native token) is transferred into an escrow associated with that shift. The escrow logic ensures neither party can access the funds until predefined conditions are met, effectively **removing the need to trust a middleman with payments** [2]. For example, the contract might require that both the restaurant and chef call a `confirmCompletion(shiftId)` function after the shift's end time to indicate the work was done satisfactorily. Once confirmations are received (or if a configurable time window passes without any dispute filed), the contract automatically allows the chef to withdraw the funds. If there's a dispute (e.g., a party does not confirm or explicitly flags an issue), the funds remain locked and a dispute resolution process can be triggered (this could involve an arbitration mechanism or a delay allowing off-chain resolution, depending on the platform's policy). Thanks to the deterministic smart contract rules, **funds are only remitted when conditions are objectively satisfied**, providing assurance that chefs get paid for completed work and restaurants won't pay for incomplete service [2]. The escrow contract could also implement platform fees or commissions – for instance, releasing 95% to the chef and 5% to a platform treasury address – though such details can be configured per the business model.

- **Reputation and Feedback Contract:** Trust between pseudonymous users is bootstrapped by a reputation system recorded on-chain. ChefsPlan's reputation contract tracks ratings and verifiable feedback for chefs and restaurants after each completed shift. After completion, both parties can submit a rating (e.g., 1-5 stars) or a short review, which the contract may store as an event or a numeric reputation score update. To preserve privacy and prevent trivial linking of reviews to identities, the system can aggregate scores or use cryptographic commitment schemes for detailed feedback, only surfacing an overall rating. In its simplest form, the reputation contract maintains a score or rating count for each user address (chef or restaurant) that updates when new reviews are submitted [3]. Because this reputation data lives on-chain, it

is tamper-proof and transparently auditable by users or third-party services. A **Reputation System Contract** can automatically track how trustworthy a user is by aggregating their history of fulfilling agreements and feedback from counterparts [3]. Good behavior (completing shifts successfully, getting good reviews) raises a user's score, while bad behavior (no-shows, cancellations, poor reviews) can lower it. The rules for how scores change can be coded (for example, a positive review adds points, a no-show might subtract a larger penalty, etc.), potentially with community governance to adjust the formula if needed. This on-chain reputation provides a decentralized way to evaluate credibility without relying on a single company's database – it "helps ensure users can trust each other in decentralized apps by providing a clear and fair way to evaluate credibility" [3]. Importantly, because it's on zkSync, updating and reading these scores has low gas cost, enabling frequent updates without burdening users.

- **Smart Contract Patterns and Security:** The smart contracts are designed following established patterns for security and upgradeability. For instance, the escrow and shift management could be implemented in a single contract with distinct functions, or as a set of interconnected contracts (one handling shifts and applications, another handling payments) depending on gas efficiency and clarity. To allow iterative improvements, ChefsPlan might use the proxy pattern (UUPS or Transparent proxies) for upgradeable contracts, governed by a multi-sig or DAO representing the platform stakeholders. All contracts are audited and make extensive use of checks-effects-interactions pattern to avoid reentrancy issues (especially in escrow payouts). On zkSync, the account model and **native account abstraction** allow features like contract-based accounts and meta-transactions. This means ChefsPlan could let users sign a transaction that an agent or relayer submits and even sponsor transaction fees for first-time users, smoothing the onboarding UX. The choice of zkSync ensures that every state update (escrow deposit, reputation change, etc.) is eventually backed by a zero-knowledge proof, guaranteeing the correctness of the L2 state changes when they are finalized on Ethereum [4]. Overall, the on-chain architecture enforces that **agreements and payments are executed exactly as coded** – participants do not need to trust a company's promises, only the transparent logic of the smart contracts.

Pseudocode snippet – ChefsPlan core contract interface: Below is a conceptual excerpt of how the interface of ChefsPlan's smart contracts might look (for illustration):

```
contract ChefsPlanPlatform {
    enum ShiftStatus { Open, Accepted, Completed, Cancelled, Disputed }

    struct Shift {
       address restaurant;
       address chef;
       uint256 paymentAmount;
       string detailsCID;    // IPFS CID for job description
       ShiftStatus status;
    }

    // Post a new shift (restaurant creates a job listing)
    function postShift(uint256 paymentAmount, string calldata detailsCID)
        external returns (uint256 shiftId);

    // Chef applies to an open shift (could emit an event or mark interest)
    function applyToShift(uint256 shiftId) external;
```

```
    // Restaurant accepts a chef for the shift and funds escrow
    function acceptApplicant(uint256 shiftId, address chef) external payable;

    // Both parties confirm completion of the shift
    function confirmCompletion(uint256 shiftId) external;

    // (Optionally) raise a dispute
    function raiseDispute(uint256 shiftId, string calldata reason) external;

    // Withdraw payout if shift completed and caller is the chef
    function withdrawPayout(uint256 shiftId) external;

    // Reputation: submit a rating after completion
    function rateCounterparty(uint256 shiftId, uint8 rating) external;

    // Events for monitoring off-chain:
    event ShiftPosted(uint256 indexed shiftId, address indexed restaurant);
    event ShiftAccepted(uint256 indexed shiftId, address indexed chef);
    event ShiftCompleted(uint256 indexed shiftId);
    event ShiftCancelled(uint256 indexed shiftId);
}
```

Explanation: Restaurants use `postShift` to create a listing with a payment offer and an IPFS hash of details. Chefs call `applyToShift` to signal interest (the contract might record the applicant list or simply emit an event for off-chain tracking to pick up). The restaurant then calls `acceptApplicant` with the chosen chef's address and attaches the payment amount, moving funds into escrow under the contract. The shift's status becomes Accepted and is linked to that chef. After the work date, both call `confirmCompletion` which marks the shift Completed if both do so (or one side's confirmation could suffice after a timeout if no dispute). Now the chef can call `withdrawPayout` to release the escrowed funds to their wallet. If something goes wrong, `raiseDispute` can freeze the state for off-chain arbitration. Finally, `rateCounterparty` allows each side to record a rating (which the contract could log and use to update reputation scores). This interface illustrates how the **shift lifecycle** is managed on-chain, with transparency and enforceability at each step.

Decentralized Storage: IPFS Integration

To maintain decentralization without overloading the blockchain, ChefsPlan uses the **InterPlanetary File System (IPFS)** for storing all large or detailed data, including job descriptions, contracts, and activity logs. Smart contracts are very expensive for storing or retrieving bulk data, so the best practice is to keep only small identifiers or hashes on-chain and put the substantive data in a distributed storage network [1]. IPFS provides content-addressed storage: each file (or piece of data) is identified by a cryptographic hash (CID). In ChefsPlan, whenever a restaurant posts a new shift, the full description of that shift (role requirements, location, schedule, any attached documents like menus or kitchen guidelines) is uploaded to IPFS. The platform might do this via an IPFS API or through a node, and it obtains the CID hash. That CID is then stored in the Shift contract on-chain as part of the shift struct (`detailsCID` in the pseudocode above). This way, anyone with the CID can fetch the data from IPFS, but the on-chain record remains minimal (just a hash string).

Job postings: By clicking on a shift listing in the dApp, a chef's frontend will use the CID from the blockchain to retrieve the job description from IPFS. Because IPFS is peer-to-peer and content-

addressed, the chef can be confident that the data is exactly what the restaurant uploaded (any tampering would change the hash). Storing these descriptions off-chain dramatically reduces gas costs and keeps the blockchain lean, while IPFS ensures the data is **distributed and immutable** (content cannot be altered without changing the hash reference).

Documents and logs: ChefsPlan may involve other documents, such as copies of a chef's certifications, KYC documents (IDs, work permits), contracts or agreements signed by the parties, and logs or timesheets for the shifts. All such files are stored on IPFS (possibly encrypted if they are sensitive, more on privacy below). For example, during user registration or verification, a chef might upload their culinary certification PDF. Instead of sending this through a centralized server, the platform could push it to IPFS and get back a CID, which is then associated with the chef's on-chain identity (perhaps stored in a separate identity contract or off-chain database with a hash on-chain). Similarly, important events or logs could be stored as JSON objects on IPFS. For instance, at the completion of a shift, the system could create a "shift completion report" JSON (containing timestamp, both parties' signatures or confirmations, hours worked, etc.), put that on IPFS, and store the hash in an event on-chain. This provides an **audit trail** that is tamper-proof (since it's hashed) yet does not bloat the blockchain. If a dispute arises, arbitrators or agents can retrieve these logs from IPFS to see what happened.

Using IPFS also improves **privacy and access control** in some cases. Data on IPFS can be encrypted so that only authorized parties (chef, restaurant, and perhaps the platform's compliance agent) have the decryption key. The IPFS hash can still be stored publicly on-chain (or shared off-chain) as evidence that a document exists and was referenced at a certain time, without revealing the content. For example, a chef's government ID scan could be encrypted and stored in IPFS with a hash on-chain; later the chef can prove via a zero-knowledge method that the ID was verified, without exposing the actual ID (see next section).

Content moderation and persistence: One challenge with IPFS is ensuring the data persists (since if no node pins it, it might not be available to others). ChefsPlan would likely maintain its own IPFS nodes or use a pinning service to guarantee that important files (like job descriptions and agreements) remain available in the network. Additionally, the platform's terms can disallow posting inappropriate content, and since everything is content-addressed, any illicit content could be identified by its hash. The use of IPFS aligns with the platform's goal of **decentralization** – it avoids central servers for file storage and instead relies on a distributed network, meaning the availability of data does not depend on a single company's uptime.

In summary, IPFS integration allows ChefsPlan to **store large and rich data off-chain while anchoring references on-chain**, achieving both efficiency and integrity. This way, the Ethereum/zkSync blockchain is used for what it's good at (financial state, small data, verification), and IPFS is used for what it excels at (large file storage and distribution).

Privacy and Compliance via Zero-Knowledge Proofs

A standout feature of ChefsPlan is its use of **zero-knowledge proofs (ZKPs)** to reconcile strict compliance requirements with user privacy. In the hospitality staffing domain, various regulations and qualifications must be verified: chefs may need food safety certifications, proof of identity/age (especially if serving alcohol), or work authorization in a given country; restaurants might need to verify their business license or compliance with labor laws. Traditionally, verifying these would require sharing sensitive personal data with the platform or each counterparty. ChefsPlan instead employs ZKPs to allow **"trust but verify"** – participants prove they meet requirements without revealing the underlying sensitive data [5] [6].

Compliance proofs: For example, consider Know-Your-Customer (KYC) and work eligibility. When a chef joins the platform, they must prove their identity has been verified (to comply with regulations and build trust). Rather than uploading their passport to a centralized server, the chef could engage with a third-party identity provider that supports zero-knowledge attestations. One approach is using something like the Self Sovereign Identity or protocols like the Self Protocol or Polygon ID, where the user obtains a ZK credential. The chef might get an attestation from a KYC provider that *"User X has a valid ID and is over 18, and not on any watchlist"*. The provider gives the chef a cryptographic proof (a ZK-SNARK or similar) that can be verified by a smart contract or off-chain agent. The proof might be as simple as proving "I am verified and my government ID age ≥ 18" without exposing name or birthdate. Vitalik Buterin and others have noted that with zk-SNARKs one can prove attributes like being over a certain age without revealing the actual date of birth [5]. In ChefsPlan, the **Compliance Agent** (off-chain component) or a smart contract can verify this proof. On verification, the chef's on-chain profile could be marked as "KYC verified" or "certified" in some way that doesn't reveal any PII (personally identifiable information). The zero-knowledge approach means that the blockchain and other users learn *nothing* beyond the fact that the proof is valid – the chef's sensitive info remains secret. Yet, the restaurant hiring can be confident (via the on-chain flag or credential token) that "this chef is verified and qualified," satisfying compliance.

Qualification proofs: Similarly, consider professional certifications or health/safety training certificates. The platform can require chefs to hold certain certificates (e.g., a hygiene certificate). A certifying authority (say a government or training body) could issue a digital credential. Instead of posting the certificate publicly, the chef uses a ZKP to prove to the Compliance Agent or a verification contract that "I hold a valid certificate X" or even "I scored above threshold in test Y" without revealing the certificate itself. This could be done by encoding the certificate details into a hash and having the authority sign it; the chef then proves knowledge of a valid signature meeting criteria, *without revealing the actual document*. These techniques ensure compliance checks are done **without violating user privacy**. The outcome might be an on-chain NFT or soulbound token representing the qualification, but minted in such a way that it doesn't disclose more than necessary (perhaps just an ID number or an anonymous credential record).

ZK KYC on zkSync: zkSync's ecosystem is particularly suited for ZK identity solutions. In fact, zkSync is designed with features for identity and selective disclosure built-in, allowing one to meet KYC/AML needs "without exposing competitive data" [7]. Projects like RNS.id and QuarkID have integrated with zkSync to provide on-chain KYC where the user's data is hashed or kept off-chain, and only proofs are put on-chain [8]. ChefsPlan can leverage such integrations. For example, a restaurant could be required to prove they have a valid business registration. The platform might use the Dutch KvK (Chamber of Commerce) API off-chain to check a business, but instead of storing the business details on-chain, it could issue a ZK proof that "Venue with KvK #XYZ is a registered business in good standing" which the restaurant's address then associates with. The **Compliance Agent** might handle gathering these proofs (interacting with external systems and producing or verifying proofs), whereas the blockchain stores only minimal attestations (like a boolean flag or a hash of the verified attribute).

Using ZKPs, **regulatory compliance is achieved in a *provable* yet privacy-preserving way**. This increases trust: a restaurant knows the chef is qualified (so liability is covered) and a chef knows the restaurant is legitimate, all without exchanging paper IDs over email. Moreover, from a data security standpoint, there's no honeypot of personal data to be hacked – sensitive information stays off-chain and only proofs are shared. As John Izaguirre notes, *"ZK doesn't erase compliance; it actually helps: prove KYC passed without exposing documents."* [9]. That principle guides ChefsPlan's design.

On-chain verification vs off-chain: Depending on complexity, some proofs might be verified directly in a smart contract (if the proof is succinct and verification is efficient enough) or by an off-chain service.

For instance, zkSync Era supports verifying certain SNARK proofs on-chain. For heavier computations (like verifying an official government certificate's digital signature), the Compliance Agent could do it off-chain and then update a contract state (this introduces a slight trust assumption in the agent's honesty, but the agent's actions can be audited or bound by stake). Over time, as ZK technology progresses, more of these verifications could move on-chain completely.

Selective disclosure and privacy: Beyond compliance, ZKPs can enhance user privacy in other ways on the platform. Users might prove reputation or history without revealing full details. For example, a chef could prove "I have completed more than 10 shifts with an average rating above 4" with a ZK proof that aggregates their history, instead of the contract needing to reveal each past rating publicly. This could mitigate issues like very detailed work history being exposed (which might deanonymize users or expose sensitive patterns). If ChefsPlan issues soulbound tokens for achievements, ZK can allow proving possession of certain badges without linking them directly to the user's main account, if anonymity is desired. These features would be part of a later phase, but the whitepaper highlights them to show the commitment to privacy.

In summary, **zero-knowledge proofs imbue ChefsPlan with strong privacy and trust guarantees**: participants can comply with regulations and build trust ("I am qualified and reputable") without unnecessary data exposure. This fosters a more open ecosystem where users maintain control of their personal information. The combination of zkSync's scalability and ZK privacy techniques helps ChefsPlan achieve a balance of transparency and confidentiality that traditional systems cannot easily offer.

Off-Chain Orchestration via Autonomous Agents

While the smart contracts and IPFS provide the decentralized backbone, ChefsPlan employs a suite of **off-chain autonomous agents** to handle coordination, business logic that doesn't need to be on-chain, and integration with external systems. These agents run as background services (which could be centralized servers initially, but in the future could be decentralized or community-run processes). They work by listening to blockchain events, responding to off-chain triggers, and sometimes initiating on-chain transactions. Here we describe the core agents: Matching Agent, Compliance Agent, Payout Agent, and Copilot.

- **Matching Agent:** The Matching Agent is responsible for optimizing the pairing of chefs with open shifts. It continuously monitors the pool of open shift postings (e.g., by reading events from the Shift contract or querying an indexed database of current open shifts) and the pool of available/active chefs. Using criteria like skill match, location proximity, chef's availability calendar, and reputation, it can suggest potential matches. For instance, when a new shift is posted on-chain, the Matching Agent might automatically identify a list of qualified chefs (perhaps those who have indicated they are actively looking for gigs in that area) and send them notifications off-chain. These notifications could be push alerts through the app or even automated emails, encouraging chefs to apply. The agent could also do the reverse: if a chef updates their profile or availability, it might inform relevant restaurants or flag the chef in a search index. In future iterations, the Matching Agent might employ AI/ML techniques to improve match quality (learning which matches tend to result in successful gigs) – but always as a recommendation system. Importantly, **all final decisions (accepting a match)** still happen via on-chain actions by the users; the Matching Agent simply helps automate the discovery process, acting as a decentralized "matchmaker" service.

- **Compliance Agent:** As discussed in the ZK section, the Compliance Agent handles verifying user credentials and enforcing any business rules that span on-chain and off-chain. This agent

interfaces with external data sources such as government databases, sanction lists, or certification registries. For example, if a chef claims a certain certification, the agent might verify it through an API (or check an attestation on-chain from a recognized issuer). It can then either issue a zero-knowledge proof or update a smart contract with the result. The Compliance Agent also ensures ongoing compliance: it might run periodic checks (say, confirming that a restaurant's business license hasn't expired, or that a chef's work permit is still valid each year). If something becomes non-compliant, the agent could flag the user account (perhaps pausing new hires until resolved). In terms of privacy, this agent is designed to handle data carefully – e.g., it might fetch a document from IPFS, decrypt it to verify something, but never store the plaintext, only keep a proof or outcome. The Compliance Agent essentially bridges the gap between **real-world legal requirements** and the on-chain platform, automating tasks that would otherwise require manual admin work. By doing so as an autonomous agent, it reduces the need to trust a human intermediary for checks, and it can operate 24/7 reacting immediately as needed.

- **Payout Agent:** The Payout Agent's role is to ensure smooth financial flows, especially in scenarios where automated execution needs a little off-chain help. In an ideal scenario, once a shift is marked completed on-chain, the escrow contract allows withdrawal. However, there could be situations where neither party triggers the payout (for instance, the chef might not know they need to call `withdrawPayout` or their wallet might be offline). The Payout Agent monitors for *completed shift events* or for escrow states that are ready for release. If, say, 24 hours pass after both parties confirmed completion and the money hasn't been claimed, the Payout Agent can step in to call the contract's payout function on behalf of the chef (this could be pre-authorized or done via a meta-transaction if the contract allows). This ensures timely payment. The agent can also handle edge cases: if a dispute was raised, it might notify human arbitrators or follow a resolution protocol; if the dispute is resolved off-chain (e.g., an arbitrator signals a winner), the agent could feed that result into the smart contract (perhaps via a privileged call or oracle mechanism) to release funds accordingly. Essentially, the Payout Agent acts as an **automation layer for settlement**, ensuring no funds get stuck and that all parties receive money as soon as conditions allow. It might also handle conversion or bridging if needed (for example, if users want payouts on a different chain or in fiat, the agent could coordinate with a payment gateway off-chain, but that's an extension beyond core).

- **AI Copilot Agent:** The Copilot is an AI-driven autonomous agent that contributes to platform management and user assistance. Internally, the Copilot monitors the overall system health and performance. It can watch the other agents and services, detect anomalies or inefficiencies, and even attempt self-healing actions. For instance, if the Matching Agent fails to send a notification, the Copilot might detect the error and restart that process or alert a developer. The Copilot might also use AI to analyze usage patterns and suggest improvements (like identifying peak times for shift postings and advising the system to allocate more resources then). For developers and administrators, the Copilot provides an automated "ops assistant" – e.g., generating updated documentation of the system state, running test scenarios, and ensuring the platform's smart contracts and agents remain in sync (this mirrors the **monitoring system described in the ChefsPlan development docs**, where an agent updates PLAN.md and runs tests [10] [11]). On the user-facing side, the Copilot could eventually serve as a chatbot or recommendation engine: it could help a new user by walking them through the process of posting a shift or applying for one, or even auto-suggesting improvements to a job post (like "your offered rate is below the average for this role"). Since it's AI-driven, it can continuously learn from interactions. However, all its suggestions remain within the policies set by the platform – it won't execute critical actions without approval, but it will make the system more responsive and smart. In essence, the AI Copilot ties everything together, **orchestrating** the agents and providing intelligent assistance to make the user experience smoother and the system maintenance easier.

These off-chain agents communicate with the blockchain via RPC calls and can also talk to each other as needed. Security is paramount: for example, if any agent has the ability to trigger a contract function (like the Payout Agent releasing funds), it must be properly authenticated and ideally governed (one could implement that only a specific agent address or oracle can call a certain contract method). In early versions, these agents might be centralized services run by the ChefsPlan team for simplicity, but the architecture allows them to be distributed or even open-source so the community could run watchers that verify the main agent's actions. By modularizing these concerns into separate agents, ChefsPlan achieves a flexible design where each component does one thing well – matching, compliance, payouts, support – and can be improved or scaled independently. This multi-agent system ensures that **off-chain processes are automated**, reliable, and can operate continuously without intensive manual oversight, complementing the on-chain logic.

Trust and Privacy Guarantees

ChefsPlan is built to maximize trust and privacy for all participants, leveraging its architectural choices to provide strong guarantees:

Trustless Transactions: By using blockchain smart contracts for core transactions, the platform removes the need to trust a central party for critical operations. Payments are secured in escrow smart contracts which neither the chef nor the restaurant (nor even the platform owners) can unilaterally seize or redirect. Funds are held and released strictly according to the coded conditions, backed by the security of Ethereum's network [2]. This dramatically reduces counterparty risk: a chef doesn't have to worry about a restaurant failing to pay an invoice – the payment is pre-funded and locked on-chain. Likewise, a restaurant knows that it will only pay if the service is delivered, or else can get a refund, without needing to chase a refund through customer support. The elimination of a traditional escrow agent or financial intermediary also cuts out high fees and delays, as the API3 escrow analysis highlighted (bank wire holds, working hours constraints, etc.) [12] [13]. Trust is instead placed in open-source code and math. The **immutability** of smart contracts means the rules of the game cannot be suddenly changed or selectively applied – all users operate under the same transparent conditions.

Decentralized Reputation and Identity: The on-chain reputation system provides a tamper-proof record of user trustworthiness. Because reputation entries are stored on the blockchain, no central administrator can secretly alter a user's ratings or delete negative feedback. This protects against censorship or favoritism. Users can inspect the history (to the extent it's public) and even outside developers could build reputation dashboards or analytics since the data is openly accessible. Moreover, since reputation is tied to blockchain addresses, a user truly "owns" their reputation in a way – if ChefsPlan were to ever become interoperable with other platforms (or if it became community-run), that reputation can carry over as a verifiable asset. The use of **soulbound tokens** or similar non-transferable badges for completed gigs could further reinforce trust: for example, a chef with 50 completed shift badges has a publicly verifiable work history. Privacy is preserved by not exposing more data than necessary – e.g., the system might show aggregate ratings but not the exact details of each review on-chain, keeping detailed feedback in IPFS or off-chain to avoid reputational data being used maliciously by unrelated parties.

Selective Privacy via ZKPs: ChefsPlan employs zero-knowledge proofs to strike a balance between transparency and confidentiality. Users and their data are protected: sensitive information (identity documents, certifications, contact details) are never broadcast on the public blockchain. Instead, the blockchain holds attestations or hashed identifiers. For instance, a chef's real name or government ID number is not on-chain, but a proof that "this address is owned by a verified individual" is on-chain. This means **pseudonymity** can be maintained – a chef could operate under a screen name, yet restaurants

see a “Verified” badge that was obtained through ZK proof. Likewise, details like exact home addresses or bank info are kept off-chain. Even when ZK proofs are used to verify attributes (like age or qualification), they reveal nothing except the validity of the claim. This ensures compliance with privacy laws (GDPR etc.) because personal data isn’t floating around publicly.

Data Encryption and Access Control: Content stored on IPFS can be encrypted. ChefsPlan can ensure that, for example, any legal contracts or personal documents are encrypted with keys that only the intended parties (and perhaps the Compliance Agent) possess. This way, even though IPFS is public, the content is unreadable to unauthorized viewers. From a user’s perspective, they trust that their personal documents will not leak or be misused; only cryptographic hashes or proofs derived from them are shared. Additionally, user communications (like chat between a chef and restaurant about job details) could be end-to-end encrypted and stored in IPFS or a messaging layer, so that neither the platform nor outsiders can snoop – only the participants can decrypt their messages or logs.

Autonomy and Governance: Over time, trust in the platform can be further enhanced by decentralizing governance. The roadmap includes a phase where ChefsPlan could introduce community governance (perhaps via a token or a DAO) to oversee upgrades to smart contracts or changes in fee structures. This means users collectively have a say, rather than having to trust a single company’s decisions. Even in the current design, critical actions are transparent: if the ChefsPlan core team needed to intervene (say to pause a contract in an emergency), those powers would be explicitly defined and visible in the smart contract code or multi-sig setup.

No Single Point of Failure: Because of decentralization, the platform is resilient. There is no single server that if hacked or down could stop the whole service – the blockchain and IPFS are distributed by nature. Users can always directly interact with the smart contracts via blockchain explorers or their own wallet apps if the official interface is unreachable, ensuring continuity. This **censorship resistance** builds trust that the platform will be available when needed and no user can be arbitrarily locked out as long as they have control of their crypto keys.

In sum, ChefsPlan’s design yields a system where **users trust the technology, not an intermediary**. Chefs and restaurants can engage in work agreements with confidence that: (1) money is handled fairly and automatically, (2) each party’s identity and credentials have been vetted without revealing private info, (3) reputation metrics are reliable and not manipulable, and (4) their data is in their control. By combining blockchain’s transparency with zero-knowledge privacy techniques, ChefsPlan offers what could be called *transparent fairness with confidential details*. This gives it an edge over both purely centralized platforms (which ask for a lot of trust) and naive decentralized ones (which might expose too much data). The **trust and privacy guarantees** are core to fostering a vibrant community on the platform, where users feel safe and empowered.

Key Data Objects and Structures

To better understand the ChefsPlan system, here are the key data objects and their roles, spanning both on-chain and off-chain representation:

- **Chef (User Profile):** A chef is a worker seeking gigs. On-chain, a chef is identified by their wallet address (which may be an Externally Owned Account or a smart contract account on zkSync). The chef’s profile includes data like their experience, skills, and reputation score. Off-chain, additional profile info (name, bio, certificates) may be stored in IPFS. Each chef can have associated credentials (verified ID, skill badges) represented as on-chain flags or tokens. The chef’s availability could be managed off-chain or via a separate scheduling contract (for example,

availability calendars might be kept in a centralized database or a Layer-2 storage solution for efficiency, with hashes anchored on-chain periodically).

- **Restaurant (Venue Profile):** A restaurant (or any hospitality venue) is a client offering gigs. Like chefs, they are identified on-chain by an address (which could be controlled by an organization's account). Their profile has info such as business name, location, and ratings from chefs. They might also have compliance verifications (e.g., business registration verified) recorded. The platform might enforce role-based permissions, e.g., marking an account as a "Restaurant" type after certain verification. Key data includes their posted shifts and history of fulfilled gigs, which contribute to a reputation score as well (chefs rate the restaurants too).

- **Shift:** A shift is the core gig unit – essentially a job posting or an engagement for a specific time. It typically includes details like job role (e.g., line cook for an evening), date/time, duration, pay rate or total pay, location (which could be generalized or hidden until acceptance for privacy), and requirements (like skills needed). Technically, a Shift is represented on-chain once posted: it can be a struct in a mapping (keyed by shift ID) within the Shift contract. Its fields include references to the poster (restaurant) and eventually the hired chef, the payment amount, current status, and the IPFS CID for the full description. A shift goes through statuses: *Open* (posted, accepting applications), *Accepted* (chef selected and escrow funded), *Completed* (work done and confirmed), or *Cancelled/Disputed*. Each shift has an associated escrow entry (either implicit within it or via an ID to the escrow contract). Off-chain, shifts might also appear in search indexes and geolocation services (for example, to filter shifts by distance to the chef). In many ways, a Shift is analogous to a "listing" in a marketplace and an "order" once accepted – ChefsPlan combines both into the same object for a seamless workflow.

- **Gig/Engagement:** We use "gig" often as a synonym for an accepted shift – essentially the commitment between a particular chef and restaurant. In data terms, once a shift is accepted, it becomes a specific Gig instance (with 1-to-1 mapping to the shift entry). The Gig is characterized by an agreement (possibly a signed digital contract in IPFS) between the two parties. It inherits details from the shift (like timing and pay) and fixes the participant identities. The Gig might not be a separate on-chain object from the Shift; it's just the shift entry in Accepted status with a designated chef. However, in some models, one could have an *Agreement contract or record* separate from the listing – for example, the platform might deploy a minimal **escrow contract instance for each gig** (possibly using a factory and minimal proxy pattern, so each gig has its own escrow address holding the funds and with functions for confirmation). This would isolate each gig's funds and logic, at the cost of more deployment overhead. The current design leans to a simpler single contract managing multiple gigs, but it's an adjustable detail.

- **Escrow:** The Escrow is a crucial data structure linking payment to the gig's outcome. It contains fields such as payer (restaurant), payee (chef, initially unset until acceptance), amount (in specific token), and release conditions (which could be an enum or flags indicating if each party confirmed, or a deadline for auto-release). If implemented as a distinct contract per gig, it's an address with a balance and functions; if implemented in one contract, it could be a struct in a mapping like `escrows[shiftId]`. The escrow essentially mirrors the state of the shift/gig from a financial perspective: e.g., `escrow.status = FUNDED` or `RELEASED`. It might also log a reference to any dispute or resolution outcome. Once an escrow is released or refunded, it's closed (to prevent double spending). The **security** of the escrow is underpinned by the blockchain – it is only controllable by the logic that both parties agreed to by using the platform. There is no scenario where the escrowed funds can be taken by anyone else or for any other reason than what's coded (even platform admins can't just grab it without following the contract's rules).

- **Proof (Credential Proof):** In the context of ChefsPlan, a Proof refers to a zero-knowledge proof or cryptographic attestation that a certain statement is true about a user. These proofs are not long-lived objects on-chain in the way a shift or escrow is, but rather ephemeral data that gets verified and then perhaps logged as an event or causes a state change (like setting a "verified" status). For example, a proof that "Chef Alice is over 18 and has a Food Safety Certificate" might be submitted to the Compliance Agent. The proof itself could be a blob of data (like a zk-SNARK proof bytes and associated public inputs) that the verifier checks. Once verified, the outcome is recorded – e.g., the chef's address is added to a list of certified chefs on-chain. One can consider that list or flag as a *result* of the proof. In some cases, the proof could be interactive or off-chain (like the agent does the verifying), but any on-chain verification might involve a **Verifier contract** that holds a verifying key for certain proofs. For instance, if using something like Semaphore or another ZK protocol, there might be a contract that can verify those proofs given the correct inputs. So, one of the data objects might be an **Identity Credential** record: not storing the proof itself permanently, but the fact that a user provided a valid proof at some time (with maybe a proof ID or hash). In summary, "Proof" is a transient object that enables trusted updates to user attributes without revealing secrets.

- **Reputation Record:** The reputation system will accumulate data points like ratings and feedback comments. On-chain, the simplest representation is a mapping from user address to a numeric reputation score (e.g., an integer that goes up or down) and possibly counts like number of reviews. A more complex representation could involve a struct storing several metrics (average rating, total jobs, positive count, negative count). Additionally, each individual feedback could be an object stored off-chain (like a JSON containing who rated whom, stars, comment, timestamp, maybe hashed and the hash stored on-chain for integrity). Another method is using events: when a rating is given, an event is emitted with details, which can be aggregated by off-chain indexers. The on-chain reputation record might then only store an aggregate score to be used in matchmaking or displayed on profiles. It is worth noting that **reputation is tied to addresses** – if a user switches addresses, their past reputation wouldn't carry over, which discourages users from discarding an identity due to bad reviews (a good thing for accountability, but it also means if their key is lost, reputation is lost unless there's a system to re-attest it). Some systems issue NFT badges per successful transaction (as a kind of granular reputation), which ChefsPlan could consider (each completed shift mints a non-transferable token for the chef and maybe for the restaurant as well). These NFTs themselves are data objects that contain metadata about the gig and can be tallied to derive reputation.

In summary, the ChefsPlan data model spans both **on-chain constructs (shifts, escrows, users as addresses, reputation mappings)** and **off-chain data (detailed profiles, documents, and proofs)**, with cryptographic links between them (hashes and proofs ensure consistency). The above objects are designed to encapsulate the key information flows in the system: Who is the user? What job is being done? Under what terms (when/where/how much)? Is the payment secure? Was everything verified and completed? And how does this affect each user's standing going forward? By clearly defining these entities, the platform achieves an organized flow of information and makes it easier to extend or modify parts of the system (for example, adding a new type of proof or a new attribute to the shift object can be done in one place). These objects also form the basis for any future analytics or AI suggestions – e.g., the Matching Agent looks at shift objects and chef profiles; the Compliance Agent checks proofs; the Payout Agent looks at escrow records.

Roadmap and Future Steps

ChefsPlan's development is planned in progressive phases to deliver a Minimum Viable Product (MVP) quickly and then enhance it with ZK features and scaling improvements:

- **Phase 1 – MVP (Minimum Viable Product):** The initial version focuses on core functionality to get the platform usable in a real setting. In this MVP, the essential smart contracts on zkSync are implemented: posting shifts, applying, accepting, and escrowed payment release with basic confirmation. The MVP may operate with a simpler trust model initially – for example, it might launch without full zero-knowledge verification, instead using a basic whitelist or manual verification for KYC while the ZK system is being built. The off-chain agents for matching and payouts will run in a basic form: Matching Agent can just notify users, Payout Agent handles automatic withdrawals after a fixed time, etc. The goal of MVP is to prove the concept in a live environment: a few restaurants and chefs using it, with funds flowing through zkSync contracts successfully. At this stage, some centralization may remain: e.g., the Compliance Agent might simply be a manual admin verifying documents, and the reputation system might be rudimentary (like just star ratings stored off-chain or in a centralized DB and periodically anchored to chain). MVP also involves building the user-facing dApp with wallet integration to interact with the L2. By the end of this phase, ChefsPlan should function as a basic decentralized gig marketplace on zkSync, with real funds in escrow smart contracts and maybe a small set of pilot users.

- **Phase 2 – Zero-Knowledge Integration:** Once the basic platform is stable, the next phase focuses on **integrating advanced privacy and compliance features using ZKPs**. In this phase, the platform rolls out zero-knowledge based KYC and credential verification. This could involve partnering with a ZK identity solution (for instance, integrating with something like Polygon ID, Sismo, or Self ID as referenced in the industry [6]). Chefs and restaurants will transition from any interim verification method to using these ZK credentials. The smart contracts or Compliance Agent will be updated to verify proofs on-chain or accept attestations from trusted ZK attesters. Additionally, privacy enhancements like selective disclosure for reputation (e.g., proving a threshold of performance without revealing all details) could be introduced. During this phase, any data that was temporarily stored centrally (such as personal info) can be migrated to the decentralized approach (IPFS + ZK proofs). The reputation system might also be moved fully on-chain with encrypted or ZK elements as needed. This is also the phase to tighten security: formal audits of the smart contracts, running testnets for the new features, and ensuring the zero-knowledge circuits (if any custom ones are written) are correct and efficient. By the end of Phase 2, ChefsPlan will offer a fully **compliance-friendly yet privacy-preserving experience** – all users are verified but without exposing data, and the platform can genuinely claim that it does not hold sensitive user info in a central server.

- **Phase 3 – Scalability and Expansion:** With functionality and privacy in place, the focus turns to scaling up the platform to support many users, many transactions, and possibly geographic or domain expansion. zkSync as an L2 provides a good base, but if the user base grows, ChefsPlan might consider additional measures: for instance, deploying the contracts on multiple L2 networks or sidechains to reach users where zkSync isn't as accessible, or even launching its own app-specific rollup (zkSync offers a Stack to deploy custom zk-rollups [14]). The architecture is modular enough to support a multi-chain scenario, with agents bridging data between chains if needed (for example, a future where there's a ChefsPlan on zkSync, on Polygon, etc., and perhaps a unified reputation that can aggregate across them via some oracle or bridging of SBT tokens). On the performance side, Phase 3 would involve optimizing the smart contracts (minimizing gas for certain operations as needed) and possibly leveraging zkSync's features like

batch transactions and account abstraction more heavily – e.g., enabling meta-transactions so users can pay gas in stablecoins or let the restaurant sponsor the chef's gas fee as an incentive.

Scalability is not only technical but also operational: this phase will likely add more automation to the off-chain agents. The Matching Agent might incorporate more sophisticated AI to handle tens of thousands of listings. The Copilot could start managing more of the platform's optimization (like dynamic pricing suggestions or detecting fraudulent behavior patterns). Another aspect is **ecosystem integration** – ChefsPlan could integrate with other services: for instance, plug into decentralized insurance or income smoothing services to offer chefs insurance (like how Temper had FreeSecurity insurance; a decentralized equivalent could be a DAO insurance pool that a chef can automatically contribute to or claim from via smart contract if a gig is canceled last-minute).

The roadmap's later stage might also include establishing a **governance token or DAO** for the community. This token could be given out as rewards for platform usage (chefs and restaurants earn tokens for completed gigs or good ratings), and those tokens allow voting on platform parameters (fees, dispute resolution policies, etc.). That would mark a shift to a community-governed network, making ChefsPlan not just decentralized in tech but also in control. With a DAO, even the off-chain agents could be operated by the community (for example, multiple community-run Matching Agents could compete or collaborate, each open-source, with the best algorithms being adopted by governance).

- **Beyond – Future Vision:** In the long term, ChefsPlan aims to be a fully decentralized protocol for gig work that can extend beyond chefs and restaurants to other gig verticals (bartenders, event staff, or even beyond hospitality). The architecture is general enough to support any scenario of short-term work agreements. Future improvements might include incorporating **verifiable credentials** in a broader sense (for example, integrating with national digital ID systems or work registries), using **Layer-3 or off-chain computation** with validity proofs to handle complex optimization (like AI scheduling) whose results are then proven correct on-chain [15], and achieving an even more trustless dispute resolution (possibly via decentralized arbitration courts or juror systems on-chain). Technologically, as zero-knowledge proof performance improves, ChefsPlan can move more of the logic on-chain in ZK circuits if desired (for instance, doing reputation score calculation in a ZK circuit to hide individual contributions but prove the final score). The ultimate goal is a **scalable, global platform** where thousands of gigs are handled daily through smart contracts, with negligible fees and quick finality, enabling the gig economy to run on decentralized rails.

Throughout these phases, continuous feedback from users (chefs and restaurants) will shape improvements. The roadmap is designed such that each phase delivers a working product increment, with increasing decentralization and capability. By the final phases, ChefsPlan would stand as a showcase of how blockchain and zero-knowledge technology can **empower a real-world labor market**, providing security and privacy for individuals in a way legacy platforms cannot – truly realizing the original vision of a decentralized gig-work economy.

Sources:

1. Infuy Tech Blog – *7 Ways to Optimize Gas in Solidity*: advocating off-chain storage for large data [1].
2. API3 (Medium) – *Redefining Trust: Smart Escrow Contracts*: on smart contracts enabling trust-minimized escrow without third-party intervention [2].
3. John Izaguirre (Medium) – *Zero-Knowledge, Real Internet*: on using ZK proofs for compliance (KYC) without exposing personal data [9].

4. Nadcab Labs – *Smart Contracts for Reputation Systems*: explaining on-chain reputation scores and trust in decentralized apps [3].

---

[1] 7 Simple Ways to Optimize Gas in Solidity Smart Contracts - Blockchain Development Company
https://www.infuy.com/blog/7-simple-ways-to-optimize-gas-in-solidity-smart-contracts/

[2] [12] [13] Redefining Trust: Smart Escrow Contracts | by Erich Dylus | Api3 | Medium
https://medium.com/api3/redefining-trust-smart-escrow-contracts-975a8ee26703

[3] Top Reputation System for Trust in Smart Contracts
https://www.nadcab.com/blog/smart-contract-reputation-systems

[4] [5] [6] [9] [15] Zero-Knowledge, Real Internet. Why ZK Proofs Are the Next Big Shift... | by John Izaguirre. | Aug, 2025 | Medium
https://medium.com/@izaguirre.john/zero-knowledge-real-internet-4f8dbede1aeb

[7] ZKsync: The Elastic Network
https://www.zksync.io/

[8] Digital identity platform integrates with zkSync for on-chain KYC
https://cointelegraph.com/news/digital-identity-platform-integrates-with-zksync-for-on-chain-kyc

[10] [11] AGENT_README.md
https://github.com/kickenV/Chefsplan-3.0/blob/c04cfe7a72a458714fee72f51b68d560785c68f8/AGENT_README.md

[14] Overview - ZKsync Docs
https://docs.zksync.io/zk-stack