PRD — AGE: Attestation-Gated Escrow for Micro-Tasks (2-Day Hackathon Build)

> This PRD is derived from your proposal and tuned for a 48-hour "build-and-demo" scope.

---

1) Purpose & One-liner

Build a **trust-minimized escrow dApp** where a client funds a micro-task and payment is released to the worker **only after a valid EAS attestation** confirms completion. If the deadline passes with no attestation, the client can refund.

- **Core primitive**: Attestation-gated payout (portable, queryable completion receipts).
- **Demo target**: Run on localhost UI, connect to **Sepolia** RPC for real transactions. Sepolia is Ethereum's primary public testnet for development. (Alchemy)
- **No custom backend** (static Next.js + wallet; contract + EAS on chain).

---

2) Users & Roles

- **Client**: Opens task escrow, funds it, receives refund if no attestation by deadline.
- **Worker**: Submits work reference (URL/hash). Receives payout when attested.
- **Attestor** (often the client; could be teammate/third-party): Issues **EAS** attestation binding to the task.

EAS is a free & open protocol for registering schemas and making on-chain attestations via **SchemaRegistry** and **EAS** contracts. (GitHub)

---

3) Success Criteria (Demo-Ready)

- Create task → fund → submit work → issue attestation → **contract auto-verifies attestation** (by UID) and releases funds.
- Refund path works after deadline without attestation.
- Clean, minimal UI with wallet connect; all flows finish in < 3 clicks each.
- Source available, with scripts to deploy contract to Sepolia and seed a test task.
- Optional: show last N attestations for an address or schema.

---

4) Scope (MVP vs Stretch)

In-scope (48h)

- Solidity escrow contract (ETH + ERC-20).
- EAS integration (verify attestation **UID** against expected schema/attestor).
- Next.js app (App Router), wallet connect, task CRUD (client-side), TX status toasts.
- Minimal indexing via **direct EAS read by UID** in the contract and read-only UI queries. EAS exposes `getAttestation(bytes32 uid)` on chain. (Ethereum (ETH) Blockchain Explorer)

Stretch (time-boxed, “if time permits”)

- **Subgraph** to index Tasks/Attestations for richer lists/analytics. (Quickstart is documented by The Graph.) (The Graph)
- Deploy also to Monad testnet (EVM-compatible, high TPS) for “scale” demo. (developers.monad.xyz)

---

5) Tech Stack & Key Choices

- **Frontend**: Next.js 14 (TS), Tailwind, shadcn/ui, **wagmi + viem**, **RainbowKit** for wallet. RainbowKit scaffolds Next.js + wagmi quickly and sits on viem. (rainbowkit.com)

- **Wallets**: WalletConnect (needs `WALLETCONNECT_PROJECT_ID`).
- **Chain & RPC**: **Sepolia** via Alchemy (free key fast). Quickstart is documented by Alchemy. (Alchemy)
- **Contracts**: Solidity **≥0.8.x** (checked arithmetic by default), OpenZeppelin **SafeERC20** utilities. (Solidity Documentation)
- **Dev**: Hardhat 3 for compile/test/deploy. (Hardhat)
- **Attestations**: **EAS** contracts/SDK; initialize with chain-specific EAS address. (Docs show addresses & SDK init pattern.) (Attest)

---

6) Information Architecture & Site Map

- **/** — Home: tagline, “Create Task” CTA, recent tasks (local state or on-chain query).
- **/task/new** — Create & fund.
- **/task/[id]** — Task detail: roles-aware actions (submit work, payout, refund), attestation status by UID.
- **/me** — “My Tasks” (as Client/Worker); filter tabs.
- **/attest** — Guided form to issue EAS attestation (or deep-link to EAS Scan if preferred).

Global UI

- Header: Connect Wallet (RainbowKit), chain pill (Sepolia), theme toggle.
- Toasts for TX lifecycle; empty states; error banners (network/account/allowance).

---

7) UX Flows (Happy Paths)

A) Client creates & funds task

1. Connect wallet → “Create Task”.

2. Inputs: `title`, `worker address`, `attestor address` (defaults to client), `token` (ETH/erc20), `amount`, `deadline`, `notes`.

3. Click **Create & Fund** → `createTask` (and `fundTask` if ERC-20) TX. Shows on /task/[id].

B) Worker submits work

1. On task page (role = worker) → paste **work URL** (e.g., GitHub Gist) and optional **content hash**.

2. Click **Submit Work** → TX.

C) Attestation-gated payout

1. Attestor issues EAS attestation for **TaskCompleted** schema (via dApp or EASScan).

2. On task page, anyone can click **Release Payment** supplying the **attestation UID** → contract validates and pays worker.

D) Refund

- After `deadline` and if not paid, client clicks **Refund** → funds return to client.

---

8) Smart Contract Spec (Solidity 0.8.x)

8.1 Storage

```
struct Task {
  address client;
  address worker;
  address attestor;
  address token;       // address(0) = ETH
  uint256 amount;
  uint64  deadline;    // unix seconds
  Status  status;
  string  workUri;     // optional
  bytes32 attestationUid; // set on payout
}

enum Status { Open, Submitted, Paid, Refunded }
```

mapping(uint256 => Task) public tasks; // taskId => Task

- **Arithmetic** relies on Solidity 0.8's default overflow/underflow checks. (Solidity Documentation)
- **ERC-20** transfers via **OpenZeppelin SafeERC20** to handle non-standard tokens. (OpenZeppelin Docs)

8.2 External Interfaces

```
interface IEAS {
  function getAttestation(bytes32 uid)
    external view
    returns (/* attestation fields incl. schema, recipient, attester, refUID, data, ... */);
}
interface ISchemaRegistry { /* optional read of schema UID */ }
```

EAS exposes `getAttestation(uid)` allowing on-chain attestation lookup/validation. (Ethereum (ETH) Blockchain Explorer)

8.3 Functions (happy-path signatures)

- `createTask(uint256 taskId, address worker, address attestor, address token, uint256 amount, uint64 deadline)`
- `fundTask(uint256 taskId)` payable (ETH) or **ERC-20** (requires pre-`approve`).
- `submitWork(uint256 taskId, string calldata workUri)`
- `releasePayment(uint256 taskId, bytes32 easUid)` → validates EAS → pays worker.
- `refund(uint256 taskId)` (post-deadline, if not `Paid`).

8.4 Events & Errors

- `event TaskCreated(uint256 id, address client, address worker, address attestor, address token, uint256 amount, uint64 deadline);`

- `event WorkSubmitted(uint256 id, string workUri);`
- `event Paid(uint256 id, bytes32 attestationUid);`
- `event Refunded(uint256 id);`
- `error NotClient(); error NotWorker(); error DeadlineNotPassed(); error InvalidAttestation(); error WrongAmount(); error BadStatus();`

8.5 EAS Validation Rules (inside `releasePayment`)

Given `bytes32 easUid`:

1. `att = EAS.getAttestation(easUid)`; require att exists. (Ethereum (ETH) Blockchain Explorer)
2. **Schema check**: `att.schema == TASK_COMPLETED_SCHEMA_UID` (constant set at deploy/config). (EAS docs show addressing schemas and SDK init.) (Attest)
3. **Attester check**: `att.attester == tasks[id].attestor`.
4. **Recipient check**: `att.recipient == tasks[id].worker`.
5. **Binding check**: decode `att.data` and require `data.taskId == id`.
6. **Not expired / not revoked** (if you include expiration/revocable in schema rules).
7. Update state, transfer funds (ETH or `SafeERC20.safeTransfer`).

8.6 Security Notes

- Reentrancy guard on payout/refund.
- Pull over push when possible; here push is OK because destination is known (worker).
- Validate ERC-20 decimals only for display (contract stores raw amounts).
- Use `unchecked` only when provably safe (gas micro-opt), else default checks. (Solidity Documentation)

---

9) EAS Schema Definition

Name: `TaskCompleted`
Schema string (Solidity types per EAS):

uint256 taskId, uint8 qualityScore, string comment, address worker, address client

- Register once via EAS SDK (front-end script or Hardhat task); store **Schema UID** in env/config.
- Initialize SDK with chain EAS address (docs provide per-chain addresses; Sepolia address available in EAS docs). (Attest)
- Attestation created by `attestor` references `worker`,`client`, and the `taskId`.

> Tip: You can also issue attestations via **EAS Scan UI** on the target network for speed. (optimism-sepolia.easscan.org)

---

10) Frontend Spec

10.1 Libraries

- **RainbowKit + wagmi + viem** for wallet & contract actions. (RainbowKit relies on wagmi/viem; wagmi provides `readContract`/`writeContract` helpers.) (rainbowkit.com)
- Minimal UI: shadcn/ui, Tailwind.

10.2 Components

- `ConnectButton` (RainbowKit).
- `TaskForm` (create/fund).
- `TaskCard` (list row).
- `TaskDetail` (role-aware actions).

- `AttestationForm` (optional in-app creation via EAS SDK).
- `TxToast` (pending/success/fail).

10.3 Key Screens (wireframe notes)

- **Create Task**: vertical form; “Token: ETH | ERC-20”; when ERC-20, show current allowance & “Approve” button; then “Create & Fund”.
- **Task Detail**:
  - Top: summary (client/worker/amount/deadline/status).
  - Middle: tabs **Activity** (events), **Work** (URL/hash), **Attestation** (UID input + “Release Payment”).
  - Actions are role-gated (detect `address == roleAddress`).
- **My Tasks**: two tabs “As Client” and “As Worker”.

10.4 Interactions (wagmi/viem)

- `readContract` for task view; `writeContract` for mutations. (Wagmi)
- For ETH funding use `value:`; for ERC-20 funding do **approve** → **fund**; show allowance.
- Network guard: if `chainId !== Sepolia`, prompt to switch.

---

11) Data Model (Frontend Types)

```
type Address = `0x${string}`;

type TaskView = {
  id: bigint;
  client: Address;
  worker: Address;
  attestor: Address;
  token: Address | null; // null => ETH
  amount: bigint;
  deadline: number; // seconds
  status: 'Open' | 'Submitted' | 'Paid' | 'Refunded';
```

```
  workUri?: string;
  attestationUid?: `0x${string}`;
};

type CreateTaskInput = Omit<TaskView, 'status'|'workUri'|'attestationUid'> & { amount: bigint
};
```

---

12) Environments & Config

- `NEXT_PUBLIC_CHAIN_ID=11155111` (Sepolia). (ChainList)
- `NEXT_PUBLIC_EAS_ADDRESS=<EAS contract>`; `NEXT_PUBLIC_SCHEMA_UID=<TaskCompleted schema UID>`; (addresses documented & retrievable). (Attest)
- `NEXT_PUBLIC_ALCHEMY_API_KEY=...` (or `ALCHEMY_HTTP_URL`).
- `NEXT_PUBLIC_WALLETCONNECT_PROJECT_ID=...`

---

13) Deployment & Dev Workflow

Contracts

- Hardhat tasks: `yarn hh deploy:sepolia` → outputs `Escrow.json` (ABI & address).
- Verify via Etherscan (optional).
- Unit tests: basic status transitions, attestation validation, refund timing.

Frontend

- `npx create @rainbow-me/rainbowkit` or add RainbowKit to Next.js app. (rainbowkit.com)
- Wire wagmi config to Sepolia + Alchemy RPC (Alchemy Quickstart). (Alchemy)
- Add contract ABI, env addresses; build & run locally.

---

14) Acceptance Criteria (binary)

- Create+Fund works for **ETH** and **ERC-20** (approve → fund).
- Worker can **Submit Work** (URI stored).
- **Release Payment** validates **EAS** attestation by **UID** against schema & attestor; transfers funds.
- **Refund** after deadline if unpaid.
- UI enforces role gating; network guard to **Sepolia**.
- Clear TX states (pending/success/fail).
- README includes one-command setup for both contract & app.

---

15) API & On-Chain Interfaces

15.1 Contract ABI (key methods)

- `createTask(...)` → emits `TaskCreated`.
- `fundTask(...)` (payable/erc20).
- `submitWork(uint256 id, string workUri)`.
- `releasePayment(uint256 id, bytes32 easUid)`.
- `refund(uint256 id)`.

15.2 EAS Integration (front-end optional)

- Initialize EAS SDK with chain address; call `eas.attest({ schema, data, recipient: worker, ...})`. (SDK init pattern documented.) (Attest)
- Or deep-link user to EASScan to create attestation (copy UID). (optimism-sepolia.easscan.org)

---

16) Copy & Micro-UX

- Empty states: “No tasks yet — create one in seconds.”
- Errors:
    - Wrong network → “Switch to Sepolia to continue.”
    - Insufficient funds/allowance → inline helper text + CTA to approve/add funds.
    - Invalid attestation UID → “UID not valid for this task/schema/attestor.”

---

17) Testing Plan

- **Unit (Hardhat)**:
    1. Create → Submit → Release (ETH & ERC-20).
    2. Refund after deadline.
    3. Invalid attestation (wrong schema/attestor/recipient/taskId) reverts.
- **Integration (manual)**:
    1. Deploy contract & register schema on Sepolia.
    2. Create+Fund, Submit, Attest (via EAS SDK or EASScan), Release.
    3. No-attestation path past deadline → Refund.
- **UI smoke**: Wallet connect, chain switch, form validations, loading/toasts.

---

18) Nice-to-Have (if extra time)

- “My Reputation”: fetch latest attestations for an address by schema (client-side) using public utilities that read EAS attestations. (Base Documentation)

- Subgraph for task & payout analytics (The Graph quickstart). (The Graph)
- Deploy to Monad testnet as “scale” demo. (developers.monad.xyz)

---

19) Risks & Mitigations

- **EAS address / schema mismatches** → keep in `.env`, surface on Settings panel; validate on app boot using SDK call. (Attest)
- **ERC-20 quirks** → always use **SafeERC20** wrappers. (OpenZeppelin Docs)
- **Tight timeline** → prioritize ETH path; ERC-20 as stretch if needed.

---

20) File/Code Skeletons (for AI codegen to “vibe-code” fast)

Contracts

- `contracts/AgeEscrow.sol`
- `script/01_deploy.ts` (Hardhat)
- `script/02_register_schema.ts` (EAS SDK)

App

- `/app/(routes)/task/new/page.tsx`
- `/app/(routes)/task/[id]/page.tsx`
- `/components/TaskForm.tsx`, `/components/TaskDetail.tsx`
- `/lib/wagmi.ts` (config Sepolia/Alchemy, RainbowKit)
- `/lib/contract.ts` (ABI, addresses)
- `/lib/eas.ts` (optional SDK helper)

---

21) Implementation Order (Hour-by-Hour Heuristic)

1. **H1–4**: Scaffold RainbowKit Next app; wagmi config; envs; header/footer. (rainbowkit.com)
2. **H5–10**: Implement `AgeEscrow.sol` + unit tests (ETH first, then ERC-20). OpenZeppelin + EAS interface. (OpenZeppelin Docs)
3. **H11–14**: Deploy to Sepolia (Hardhat). (Hardhat)
4. **H15–20**: Create Task UI + write flows (approve/fund if ERC-20). (Wagmi)
5. **H21–26**: Task detail (submit, release with UID, refund).
6. **H27–30**: Optional EAS attestation form (SDK) or link to EASScan. (Attest)
7. **H31–36**: Polish, empty states, toasts, copy.
8. **H37–44**: Manual test script runs end-to-end.
9. **H45–48**: README + demo script + (optional) stretch.

---

22) Demo Script (5 minutes)

1. Connect wallet (Sepolia) → **Create & Fund** task (0.01 ETH).
2. Switch to worker account → **Submit Work** (GitHub gist).
3. As attestor (or client), issue **EAS attestation** (EASScan) → copy **UID** → Release Payment → success toast. (optimism-sepolia.easscan.org)
4. Show task marked **Paid**; worker balance increased.

Appendix — Key References

- **EAS** overview & contracts (SchemaRegistry + EAS.sol). (GitHub)
- **EAS SDK init & addresses** (initialize with chain EAS address). (Attest)

- **EAS on-chain API** (`getAttestation(uid)`; validate existence). (Ethereum (ETH) Blockchain Explorer)
- **RainbowKit / wagmi / viem** quickstart & actions. (rainbowkit.com)
- **Sepolia** testnet overview & RPC access. (Alchemy)
- **Solidity 0.8** checked arithmetic (no SafeMath needed). (Solidity Documentation)
- **OpenZeppelin SafeERC20** utilities. (OpenZeppelin Docs)
- **Hardhat** getting started. (Hardhat)
- **The Graph** subgraph quickstart (stretch). (The Graph)
- **Monad** EVM-compatible testnet (stretch). (developers.monad.xyz)

If you want, I can also drop in the exact **Solidity contract stub**, **wagmi config**, and **EAS schema registration script** next so your dev can paste-and-go.