🦀 Assignment: Deterministic Deposit Proxies on Sepolia (Rust Backend)

🎯 Objective

Build a small system that:

- Uses a **Rust backend** to precompute and deploy **CREATE2 deposit proxies** that forward value to a **FundRouter**.
- Shows the **next deposit address** in a simple front-end table.
- Monitors deposit addresses for ETH on **Sepolia**, and when funded, routes to a **treasury**.

You'll be given Solidity skeletons with a few TODOs. Complete those TODOs and wire a minimal full-stack application with a **Rust backend** and a **React (or Next.js)** frontend.

📦 What's Provided

- `contracts/DeterministicProxyDeployer.sol` — *(TODO: proxy init-code)*
- `contracts/FundRouter.sol` — *(TODOs: storage checks + ERC20 transfer)*
- `contracts/FundRouterStorage.sol` — *(ready)*
- `contracts/IFundRouter.sol` — *(ready)*

🚀 Deliverables

1. **Rust backend** providing two APIs:

- `/deposit` → Generate and return the next deterministic deposit address (CREATE2).
  - Must use the same CREATE2 formula as the Solidity deployer.
  - Store each address in a local table or file (SQLite / Postgres / JSON).
- `/router` → When called, route all ETH from stored deposit addresses to a fixed treasury address.

2. Working **contracts deployed on Sepolia**.
3. A minimal **React (or Next)** front-end with one page:

#	Deposit Address	Status	Last Balance	Actions (Deploy, Route)

   - Button: **"Get next deposit address"**
   - Button: **"Route funds to treasury"**
   - Periodic monitor loop that checks balances and updates status.
4. Simple Rust scripts or binaries:
   - `deploy_contracts.rs` → deploy Storage → Router → Deployer, print addresses.
   - `precompute.rs` → given salt, print expected proxy address.
   - `monitor.rs` → poll balances (optional; front-end can do this instead).
5. A **README.md** describing:
   - How to run backend and frontend
   - Your assumptions
   - Where you implemented each TODO

🧰 Environment & Tooling

Prerequisites

- Rust 1.80+
- Cargo
- Node 18+

- pnpm / yarn / npm
- Hardhat (or Foundry; assume Hardhat for contract part)
- Sepolia RPC (Infura / Alchemy / Ankr are fine)
- A funded Sepolia private key (ask for ETH if needed)

📁 Suggested Repo Layout

```
.
├─ contracts/                  # Solidity contracts
├─ rust-backend/               # Rust backend (Axum or Actix)
|   ├─ src/
|   ├─ Cargo.toml
|   └─ .env
├─ app/                        # React/Next.js frontend
├─ scripts/                    # optional TypeScript deploy/monitor
scripts
├─ hardhat.config.ts
└─ README.md
```

⚙️ Example `.env`

```
SEPOLIA_RPC_URL=https://sepolia.infura.io/v3/<your-key>
PRIVATE_KEY=0xabc...          # deployer key (Sepolia)
TREASURY_ADDRESS=0x123...     # where routed funds go
DATABASE_URL=sqlite://data.db
```

🦀 Rust Backend (Example)

Use **Axum** or **Actix-web**.

Endpoints

- **POST /deposit**

```
{
  "user": "0xUserAddress"
}
```

  Response:

```
{
  "deposit_address": "0xABC...",
  "salt": "0x123...",
  "note": "Send Sepolia ETH to this address."
}
```

- **POST /router**
  - Routes all deposit addresses' ETH to the treasury.
  - Response:

```
{
  "checked": 12,
  "routed": 3,
  "tx_hashes": ["0x...", "0x..."]
}
```

Behavior

- `/deposit`:
  - Generate deterministic CREATE2 address using Rust-side formula:

```
keccak256(0xff ++ deployer_address ++ salt ++
keccak256(init_code))[12..]
```

  - Store address + salt in DB or JSON.
- `/router`:
  - Query all stored deposit addresses.

- For each address with balance > 0, call your Solidity deployer to deploy and route ETH to treasury.

- Use `ethers-rs` or `alloy` for RPC and contract bindings.
- Implement periodic balance polling if needed.

⚙️ Hardhat (for Solidity)

Use Hardhat only for compiling/deploying contracts.

Quick Config Example

```
import { HardhatUserConfig } from "hardhat/config";
import "@nomicfoundation/hardhat-toolbox";
import * as dotenv from "dotenv";
dotenv.config();

const config: HardhatUserConfig = {
  solidity: "0.8.20",
  networks: {
    sepolia: {
      url: process.env.SEPOLIA_RPC_URL || "",
      accounts: process.env.PRIVATE_KEY ? [process.env.PRIVATE_KEY] :
[],
    },
  },
};

export default config;
```

🧩 package.json Scripts

```
{
  "scripts": {
    "compile": "hardhat compile",
    "deploy:sepolia": "hardhat run scripts/deploy.ts --network
sepolia",
    "rust:run": "cargo run --manifest-path rust-backend/Cargo.toml",
    "rust:watch": "cargo watch -x run",
    "frontend": "pnpm --filter app dev"
  }
}
```

💡 Implementation Guidance

1️⃣ Complete the Solidity TODOs

A. Minimal Proxy Init-Code (CREATE2)

Goal:
Deploy runtime code that forwards all calls + ETH to `FUND_ROUTER_ADDRESS`.

- Use **EIP-1167** minimal proxy pattern.
- Build init-code with `abi.encodePacked(prefix, FUND_ROUTER_ADDRESS, suffix)`.
- Return it from `_proxyInitCode()`.

B. Storage Checks in `FundRouter`

Implement:

- `_isAllowedCaller(address)`
- `_isAllowedTreasury(address)`
  - via direct interface call or `staticcall + abi.decode`.

C. ERC-20 Transfer

```
require(IERC20(token).transfer(treasuryAddress, amt), "ERC20 xfer failed");
```

D. Funding Path

Proxies forward ETH to `FundRouter.receive()`.
`transferFunds(etherAmount, ...)` then sends from router to treasury.

2️⃣ Deployment Flow

1. Deploy **FundRouterStorage(owner = deployer)**.
2. `setPermissions(deployer, 0x01)` (allowed caller).
3. `setPermissions(TREASURY_ADDRESS, 0x02)` (allowed treasury).
4. Deploy **FundRouter(storageAddress)**.
5. Deploy **DeterministicProxyDeployer(fundRouterAddress)**.
6. Print and store addresses (e.g. `deployments.json`).

3️⃣ Rust Backend Responsibilities

- Keep track of salts and predicted addresses.
- Compute CREATE2 address deterministically (using same init-code).
- Call Solidity deployer contract functions for actual deployment or routing.
- Optional: use `tokio::spawn` background tasks to check balances.
- Expose REST API for frontend interaction.

4️⃣ Front-End (React/Next.js)

A minimal dashboard with:

#	Deposit Address	Status	Last Balance	Actions
1	0xabc...def	Pending	0.00	Deploy / Route

- **Get Next Deposit Address** → calls Rust `/deposit`.
- **Route to Treasury** → calls Rust `/router`.
- **Monitor** → checks balances via backend or direct RPC.

🧪 Testing on Sepolia

Manual test example:

1. Call `/deposit` → get next deposit address.
2. Send `0.001 ETH` to that address on Sepolia.
3. Observe status changing to **Funded**.
4. Call `/router` → verify treasury address receives ETH.

⚠️ Hints & Common Pitfalls

- **Salt consistency:**
  Rust and Solidity must derive salt the same way (`keccak256(salt || msg.sender)`).
- **CREATE2 formula:**

```
keccak256(0xff ++ deployer ++ salt ++ keccak256(init_code))[12..]
```

- **Network:**
  Ensure both contracts and Rust RPC use the same chain (Sepolia, chainId 11155111).
- **Permissions:**
  You'll get `NotAuthorizedCaller` if FundRouterStorage not configured correctly.

✅ Submission Checklist

- Deployed addresses (router, storage, deployer, one proxy).
- Front-end screenshot/GIF showing:
  - Predicted address
  - Deployed status
  - Funded state
  - Routed state
- Notes on Rust implementation and Solidity TODOs.
- Any assumptions or simplifications made.
- Link to GitHub repo (backend + frontend + contracts).
- Optional: Hosted demo on **Vercel**.

🌟 Optional Stretch Goals

- Display estimated gas before routing.
- Handle ERC-20 routing flow.
- Add batch deployment or batch routing.
- Add CLI (`cargo run -- deploy` / `--route`) for automation.

Support Docs & Code

contracts/DeterministicProxyDeployer.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

/// @title DeterministicProxyDeployer (skeleton)
/// @notice Deploys minimal proxies that forward to
FUND_ROUTER_ADDRESS via CREATE2.
```

```
/// @dev A few pieces are deliberately stubbed with TODOs.
contract DeterministicProxyDeployer {
    /// @dev Replace at deployment time, or make settable in a
constructor.
    address public immutable FUND_ROUTER_ADDRESS;

    error Create2Failed();
    error InvalidBytecode();

    constructor(address fundRouter) {
        require(fundRouter != address(0), "router=0");
        FUND_ROUTER_ADDRESS = fundRouter;
    }

    // ---- Bytecode helpers ----------------------------------------
-----------

    /// @notice Returns the init code used for CREATE2 deployments.
    /// @dev TODO: Candidate must implement this to return a minimal
forwarding proxy
    ///      whose *runtime* code forwards calls (and ETH) to
FUND_ROUTER_ADDRESS.
    ///      Hints welcome: EIP-1167 style or custom minimal runtime
with CALL.
    function _proxyInitCode() internal view returns (bytes memory) {
        // TODO: return init-code bytes that deploy runtime forwarding
to FUND_ROUTER_ADDRESS
        // The original snippet had a hex string with
${FUND_ROUTER_ADDRESS} spliced in.
        // For clarity here, either:
        //  - build it with abi.encodePacked(prefix,
FUND_ROUTER_ADDRESS, suffix), or
        //  - implement an EIP-1167 minimal proxy pointing at
FUND_ROUTER_ADDRESS.
        // Revert for now so it compiles.
        revert InvalidBytecode();
```

```
    }

    /// @notice Per-caller salt derivation to avoid collisions across
different users.
    /// @dev Candidates can keep this as-is or modify in place if they
justify.
    function _deriveSalt(bytes32 userSalt, address caller) internal
pure returns (bytes32) {
        return keccak256(abi.encodePacked(userSalt, caller));
    }

    // ---- Public API ----------------------------------------------
-----------

    function deployMultiple(bytes32[] calldata salts) external returns
(address[] memory addrs) {
        bytes memory bytecode = _proxyInitCode();
        addrs = new address[](salts.length);

        for (uint256 i = 0; i < salts.length; i++) {
            bytes32 salt = _deriveSalt(salts[i], msg.sender);
            address addr;
            assembly {
                // create2(value, ptr, size, salt)
                addr := create2(0, add(bytecode, 0x20),
mload(bytecode), salt)
            }
            if (addr == address(0)) revert Create2Failed();
            addrs[i] = addr;
        }
    }

    /// @notice Pure address calculation (preview) for a given list of
salts.
    /// @dev Uses CREATE2 formula with the same derived salt logic as
deployMultiple().
```

```
    function calculateDestinationAddresses(bytes32[] calldata salts)
external view returns (address[] memory out) {
        bytes memory bytecode = _proxyInitCode();
        bytes32 initCodeHash = keccak256(bytecode);
        out = new address[](salts.length);

        for (uint256 i = 0; i < salts.length; i++) {
            bytes32 salt = _deriveSalt(salts[i], msg.sender);
            bytes32 data = keccak256(
                abi.encodePacked(bytes1(0xff), address(this), salt,
initCodeHash)
            );
            out[i] = address(uint160(uint256(data)));
        }
    }
}
```

contracts/IFundRouter.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

interface IFundRouter {
    function transferFunds(
        uint256 etherAmount,
        address[] calldata tokens,
        uint256[] calldata amounts,
        address payable treasuryAddress
    ) external;
}
```

contracts/FundRouterStorage.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
```

```
/// @title FundRouterStorage (skeleton)
/// @notice Owner-controlled bitmask permissions.
contract FundRouterStorage {
    address public owner;
    mapping(address => uint8) public permissions; // bit0=caller,
bit1=treasury

    event OwnershipTransferred(address indexed oldOwner, address
indexed newOwner);
    event PermissionsSet(address indexed who, uint8 bits);

    error NotOwner();
    error ZeroAddress();

    constructor(address _owner) {
        if (_owner == address(0)) revert ZeroAddress();
        owner = _owner;
        emit OwnershipTransferred(address(0), _owner);
    }

    modifier onlyOwner() {
        if (msg.sender != owner) revert NotOwner();
        _;
    }

    function transferOwnership(address _newOwner) external onlyOwner {
        if (_newOwner == address(0)) revert ZeroAddress();
        emit OwnershipTransferred(owner, _newOwner);
        owner = _newOwner;
    }

    /// @notice Set permission bits for an address.
    function setPermissions(address who, uint8 bits) external
onlyOwner {
        permissions[who] = bits;
```

```
        emit PermissionsSet(who, bits);
    }

    function isAllowedCaller(address who) public view returns (bool) {
        return (permissions[who] & 0x01) == 0x01;
    }

    function isAllowedTreasury(address who) public view returns (bool)
{
        return (permissions[who] & 0x02) == 0x02;
    }

    function isAllowedCallerAndTreasury(address caller, address
treasury) external view returns (bool) {
        return isAllowedCaller(caller) && isAllowedTreasury(treasury);
    }
}
```

contracts/FundRouter.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "./IFundRouter.sol";

interface IERC20 {
    function transfer(address to, uint256 value) external returns
(bool);
    function balanceOf(address who) external view returns (uint256);
}

/// @title FundRouter (skeleton)
/// @notice Pull ETH held by a proxy and forward it (and optional
ERC20s) to a treasury.
/// @dev Key checks and a couple of mechanics are TODOs for the
candidate.
```

```
contract FundRouter is IFundRouter {
    error NotAuthorizedCaller();
    error TreasuryNotAllowed();
    error LengthMismatch();
    error EthSendFailed();
    error ZeroTreasury();

    /// @dev External storage contract with allowlists.
    address public immutable STORAGE;

    constructor(address storageContract) {
        require(storageContract != address(0), "storage=0");
        STORAGE = storageContract;
    }

    /// @dev Minimal interface to the storage contract.
    function _isAllowedCaller(address a) internal view returns (bool
ok) {
        // TODO: call FundRouterStorage.isAllowedCaller(a)
        // hint: (bool s, bytes memory r) =
STORAGE.staticcall(abi.encodeWithSignature("isAllowedCaller(address)",
a));
        // then decode (bool).
        // For now, pretend false to force candidate to implement.
        ok = false;
    }

    function _isAllowedTreasury(address a) internal view returns (bool
ok) {
        // TODO: call FundRouterStorage.isAllowedTreasury(a) and
return result.
        ok = false;
    }

    /// @inheritdoc IFundRouter
    function transferFunds(
```

```
        uint256 etherAmount,
        address[] calldata tokens,
        uint256[] calldata amounts,
        address payable treasuryAddress
    ) external override {
        if (treasuryAddress == address(0)) revert ZeroTreasury();

        // TODO: enforce that msg.sender is an allowed caller
        if (!_isAllowedCaller(msg.sender)) revert
NotAuthorizedCaller();

        // TODO: enforce that treasury is allowed
        if (!_isAllowedTreasury(treasuryAddress)) revert
TreasuryNotAllowed();

        if (tokens.length != amounts.length) revert LengthMismatch();

        // ---- ETH routing (from this contract's balance) -----------
-----------
        // Assumption: ETH has already been sent to this router (e.g.,
via the proxy's fallback)
        // or msg.sender has ETH and is delegatecalling; keep it
simple: just forward from here.
        if (etherAmount > 0) {
            // IMPORTANT: this assumes the ETH is already held here.
            // A minimal proxy that forwards value to this router will
land ETH here.
            (bool ok, ) = treasuryAddress.call{value: etherAmount}
("");
            if (!ok) revert EthSendFailed();
        }

        // ---- ERC20 routing (optional) -----------------------------
----------
        for (uint256 i = 0; i < tokens.length; i++) {
            address token = tokens[i];
```

```
            uint256 amt = amounts[i];
            if (amt == 0) continue;

            // TODO: implement ERC20 transfer out.
            // Choices:
            //  - If tokens sit here, do
IERC20(token).transfer(treasuryAddress, amt).
            //  - If tokens sit on msg.sender, you'd need transferFrom
and prior approval (not defined on IERC20 above).
            // Keep it simple: assume tokens are already held here.
            // For now we leave as a stub-candidate should implement.
            // e.g. require(IERC20(token).transfer(treasuryAddress,
amt), "ERC20 transfer failed");
        }
    }

    // Accept ETH so proxies can push value here.
    receive() external payable {}
}
```