

Telkes Demo Setup: Solar Device Reporting & TKL Minting

1. Choose Your Dev Environment

- Local EVM-compatible chain: Hardhat (built-in) or Ganache CLI
- Framework: Hardhat + ethers.js or Truffle + web3.js

2. Write & Deploy the EnergyRegistry Contract

Contract (Solidity):

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.0;
```

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

```
import "@openzeppelin/contracts/access/Ownable.sol";
```

```
contract TelkesToken is ERC20, Ownable {  
    constructor() ERC20("Telkes Token", "TKL") {}  
    function mint(address to, uint256 amount) external onlyOwner {  
        _mint(to, amount);  
    }  
}
```

```
contract EnergyRegistry is Ownable {  
    TelkesToken public token;  
    mapping(bytes32 => bool) public processed;  
    constructor(address tokenAddr) {  
        token = TelkesToken(tokenAddr);  
    }  
    function reportEnergy(  
        bytes32 reportId,  
        uint256 kWh,  
        bytes calldata signature,  
        address device  
    ) external {  
        require(!processed[reportId], "Already claimed");  
        processed[reportId] = true;  
        token.mint(device, kWh * 1e8);  
    }  
}
```

Deploy Script (Hardhat):

```
async function main() {  
    const [deployer] = await ethers.getSigners();  
    const Token = await ethers.getContractFactory("TelkesToken");  
    const token = await Token.deploy();  
    await token.deployed();  
    const Reg = await ethers.getContractFactory("EnergyRegistry");  
    const registry = await Reg.deploy(token.address);  
    await registry.deployed();  
    console.log("Token:", token.address, "Registry:", registry.address);  
}  
main();
```

3. Simulate a "Solar Device"

- Generate an Ethereum keypair (for device)
- Off-chain script creates reportId and signature:

```
const reportId = ethers.utils.keccak256(  
    ethers.utils.defaultAbiCoder.encode(  
        ["address", "uint256", "uint256"],
```

```
[deviceAddr, kWh, timestamp]
)
);
const signature = await wallet.signMessage(reportId);
- Send { reportId, kWh, signature, deviceAddr } to the gateway.
```

4. Gateway Batching & On-Chain Calls

Node.js script:

```
for (let rpt of reports) {
  await registry.reportEnergy(rpt.id, rpt.kWh, rpt.sig, rpt.device);
  console.log(`Minted ${rpt.kWh} TKL to ${rpt.device}`);
}
```

5. Build a Simple Dashboard

- React + ethers.js: connect to localhost Hardhat node
- Display processed reports and token balances
- "Claim" button calls reportEnergy for testing

6. Run & Demo

1. npx hardhat node
2. Deploy contracts
3. Run gateway script to simulate reports
4. Launch React app and connect via MetaMask

Optional Extras:

- Real IoT integration via MQTT from ESP32
- Meta-transactions for gas abstraction
- Deploy to Goerli for wider testing

With this demo, you have: solar-device → signed report → on-chain mint → front-end dashboard.