

Table of Contents

<i>Hackathon Brief.....</i>	<i>1</i>
<i>Link to the Faucet</i>	<i>3</i>
<i>Link to the IDE.....</i>	<i>3</i>
<i>Explorer.....</i>	<i>3</i>
<i>Link to Chat GPT conversation</i>	<i>3</i>
<i>step-by-step developer guide.....</i>	<i>4</i>
<i>Quick references (so you don't have to hunt)</i>	<i>4</i>
<i>Start of steps</i>	<i>5</i>

Hackathon Brief

Innovate for Women's Month 2025 – Empowering South African Women with Blockchain

Calling all South African Innovators! Are you a budding software engineer, product designer, or a team passionate about leveraging cutting-edge technology to create real-world impact?

This August, in celebration of Women's Month, we invite you to participate in a unique Hackathon designed to tackle pressing challenges faced by women in South Africa.

The Challenge:

Develop innovative and scalable software solutions that address a real-world problem impacting South African women, underpinned by blockchain technology.

Your solution should demonstrate how blockchain can uniquely solve the identified problem, leveraging its inherent strengths to create a tangible and positive difference.

Theme - CodeQueens

“Let your Code be your Crown”

Why Blockchain?

Blockchain technology offers a powerful toolkit for creating secure, transparent, and efficient solutions. We are particularly interested in projects that showcase how blockchain's core capabilities can empower women and contribute to a more equitable society. Consider the following strengths of blockchain and how they might apply to your solution:

- Payments and Remittances: Enabling secure, low-cost financial transactions and access to capital.
- Zero-Knowledge Proofs: Protecting sensitive personal data while verifying information.
- Traceability: Ensuring transparency and accountability in supply chains or resource allocation.
- Distributed Ledgers: Creating immutable and verifiable records for various applications.
- Voting: Facilitating secure and transparent democratic processes.
- Transparency: Fostering trust and accountability in systems and processes.
- Your Next Great Idea: We encourage creative and out-of-the-box thinking!

Leveraging BlockDAG Tools:

To maximize your impact and unlock further opportunities, we strongly encourage the use of BlockDAG tools in your projects. Projects showcasing the effective integration of BlockDAG tools will be viewed favorably during judging and will be eligible for entry into our exclusive grant program.

Potential BlockDAG Tools to Showcase:

- BlockDAG IDE: For development and coding.
- BlockDAG Explorer: For monitoring transactions and network activity.
- BlockDAG Wallet: For secure asset management.
- BlockDAG Miners: For understanding network participation (if applicable to your solution).
- BlockDAG Payment Card: For demonstrating real-world payment integrations.
- And more... Explore the BlockDAG ecosystem and integrate tools that enhance your solution's functionality and impact.

What We're Looking For:

- Innovation: A novel and creative approach to solving a real-world problem.
- Scalability: A solution with the potential to be adopted widely and impact many.
- Impact: A clear demonstration of how your solution benefits South African women.
- Blockchain Uniqueness: A compelling argument for why blockchain is the ideal technology for your solution.
- Technical Proficiency: Well-designed and robust software.

- Effective Use of BlockDAG Tools: Clear integration and demonstration of BlockDAG tools.

Join us in shaping a brighter future for South African women through the power of innovation and blockchain technology! More details on registration, workshops, and judging criteria will be announced soon. Stay tuned!

Judging rubric (share with teams)

30 % Impact on women's lives

25 % Technical execution & gas-efficiency

20 % UX / accessibility

15 % Sustainability & revenue model

10 % Alignment with CodeQueens spirit

Remember: Your code is your crown. Build boldly, ship on-chain, and wear that crown proudly! 👑

Link to the Faucet

<https://primordial.bdagscan.com/faucet>

Link to the IDE

<https://ide.primordial.bdagscan.com>

Explorer

<https://primordial.bdagscan.com>

Link to Chat GPT conversation

<https://chatgpt.com/share/68aec031-549c-8006-8982-8337d9f282a8>

step-by-step developer guide

for **CodeQueens Microgrants** (transparent stipend payouts) from scratch: VS Code, Node/Hardhat, .env, contract .sol, deployment scripts, and the frontend that talks to MetaMask + BlockDAG Primordial testnet (BDAG). I'll include exact shell commands, file/folder structure, and where to place private keys (with strong security notes). I've kept this as a single runnable workflow you can follow on **macOS or Windows**.

I used official BlockDAG, Hardhat, MetaMask, and OpenZeppelin docs as references for network params, tooling and private-key steps. Key references: BlockDAG network/faucet details, Hardhat docs & toolbox, MetaMask support for exporting/importing keys, and OpenZeppelin contracts.

Summary of what you'll create

- Local project: codequeens-microgrants/
- contracts/Microgrant.sol (Solidity contract)
- scripts/deploy.js (Hardhat deploy script)
- hardhat.config.js (network config for Primordial / BDAG)
- .env (PRIVATE_KEY, BDAG_RPC_URL) — **never commit**
- frontend/ simple HTML/JS app that connects MetaMask and calls contract
- Commands to compile, deploy, and call contract
- How to create/export/import MetaMask keys and where to paste them (with security cautions)
- How to get test BDAG from the faucet and verify on BlockDAG explorer.

Quick references (so you don't have to hunt)

- **Primordial (BlockDAG testnet)**
 - Chain ID (decimal): 1043 — hex 0x413
 - RPC URL: <https://rpc.primordial.bdagscan.com>
 - Explorer / Faucet: <https://primordial.bdagscan.com> and /faucet.
- docs.blockdagnetwork.io/primordial.bdagscan.com
- Recommended tooling:

- Node.js LTS, npm/yarn, VS Code, MetaMask (browser extension).
- Hardhat + @nomicfoundation/hardhat-toolbox for modern Hardhat setup.

Start of steps

0 — Prerequisites (macOS / Windows)

- Install Node.js LTS:
- macOS: download from nodejs.org or brew install node
- Windows: download installer from nodejs.org
- Install VS Code: <https://code.visualstudio.com>
- Install MetaMask extension in your browser (Chrome/Brave/Edge/Firefox).

1 — Create project folder & init npm

Open a terminal (macOS Terminal / Windows PowerShell or Git Bash) and run:

```
bash
```

 Copy code

```
# create project folder
mkdir codequeens-microgrants
cd codequeens-microgrants

# initialize npm
npm init -y
```

2 — Install dev & runtime packages

Run these commands in the project root.

bash

 Copy code

```
# install Hardhat + recommended toolbox (includes ethers & useful plugin)
npm install --save-dev hardhat @nomicfoundation/hardhat-toolbox
```

```
# install OpenZeppelin contracts (runtime dependency)
npm install @openzeppelin/contracts
```

```
# install dotenv to read .env
npm install dotenv
```


```
# (Optional) If you plan to write TS for scripts/frontend
npm install --save-dev typescript ts-node @types/node
```

`@nomicfoundation/hardhat-toolbox` bundles the common Hardhat plugins (ethers, waffle, etc.) and is recommended. [npm](#) [hardhat.org](#)

3 — Initialize Hardhat project

Run:

bash

 Copy code

```
npx hardhat
```

Choose **Create a basic sample project** (or JavaScript project). This will create sample folders: `contracts/`, `scripts/`, `test/`, and a `hardhat.config.js`. You'll overwrite/extend these.

4 — Project folder structure (what we'll have)

pgsql


 Copy code

```
codequeens-microgrants/  
├─ contracts/  
│   └─ Microgrant.sol  
├─ scripts/  
│   └─ deploy.js  
├─ frontend/  
│   ├── index.html  
│   └─ app.js  
├─ .env.example  
├─ .env                # NOT committed  
├─ hardhat.config.js  
└─ package.json
```

5 — Create `.env` (sensitive values) — example + use

Create `.env.example` (safe to commit):


ini

 Copy code

```
BDAG_RPC_URL="https://rpc.primordial.bdagscan.com"
PRIVATE_KEY="0xYOUR_PRIVATE_KEY_HERE" # DO NOT COMMIT .env with real k
```

Create `.env` from `.env.example` and fill `PRIVATE_KEY` (explained below). Add `.env` to `.gitignore`:

bash

 Copy code

```
# .gitignore
node_modules/
.env
.cache/
```

Important security note: Never commit `.env` or any private key into source control. Use a testnet-only MetaMask account (funded from faucet) for deployment.

6 — How to get a private key (MetaMask) — steps & safety

(1) Create/import the MetaMask account you will use for deployment or use an account with test BDAG.

(2) **Export private key** (for local script use):

- In MetaMask extension: click the account icon → *Account details* → *Export private key* → enter your MetaMask password → copy the private key.
- Or use the MetaMask support docs for exact steps. **Only use this for a test account.** Do not export mainnet / real-money account keys. support.metamask.io +1

(3) Paste into `.env` as `PRIVATE_KEY=0x...` (prefix `0x`). Hardhat scripts will pick it up.

(4) Alternative: use `mnemonic` or use `hardhat` accounts for testing (no key export needed).

7 — Hardhat config — `hardhat.config.js` example

Create/replace `hardhat.config.js` in the project root with:

```
js Copy code

require("dotenv").config();
require("@nomicfoundation/hardhat-toolbox");

const BDAG_RPC_URL = process.env.BDAG_RPC_URL || "https://rpc.primordial.bdagscan.com";
const PRIVATE_KEY = process.env.PRIVATE_KEY || "";

module.exports = {
  solidity: "0.8.19",
  networks: {
    hardhat: {
      // default local test network
    },
    primordial: {
      url: BDAG_RPC_URL,
      chainId: 1043, // BlockDAG Primordial testnet
      accounts: PRIVATE_KEY ? [PRIVATE_KEY] : [],
    },
  },
  etherscan: {
    // optional: if/when BlockDAG supports Etherscan-like verification plugin (leave blank)
  },
};
```

This config uses the Primordial RPC and chainId: 1043 (BlockDAG Primordial). Put your exported PRIVATE_KEY into .env.

8 — Smart contract: `contracts/Microgrant.sol`

Create `contracts/Microgrant.sol`:

solidity

 Copy code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

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

contract Microgrant is Ownable {
    // allocation mapping in wei
    mapping(address => uint256) public allocation;

    event Allocated(address indexed recipient, uint256 amount);
    event Paid(address indexed recipient, uint256 amount);

    // Owner can allocate an amount (in wei) to a recipient
    function allocate(address recipient, uint256 amountWei) external onlyOwner {
        require(recipient != address(0), "Invalid recipient");
        require(amountWei > 0, "Amount must be >0");
        allocation[recipient] += amountWei;
        emit Allocated(recipient, amountWei);
    }
}
```

```

// Owner can revoke allocation
function revoke(address recipient) external onlyOwner {
    allocation[recipient] = 0;
}

// Contract can receive BDAG
receive() external payable {}

// Payout allocated funds to recipient (owner initiated)
function payout(address payable recipient) external onlyOwner {
    uint256 amount = allocation[recipient];
    require(amount > 0, "No allocation");
    allocation[recipient] = 0;

    (bool success, ) = recipient.call{value: amount}("");
    require(success, "Transfer failed");
    emit Paid(recipient, amount);
}

// Helper to look up allocation
function getAllocation(address recipient) external view returns (uint256) {
    return allocation[recipient];
}
}

```

Notes:

- Amounts are in **wei**. Use `ethers.parseEther("0.01")` on the frontend/scripts to convert Ether units (BDAG uses 18 decimals).
- This is simple and intentionally minimal for hackathon MVPs.

9 — Compile & deploy script

Create `scripts/deploy.js`:

```
js Copy code

async function main() {
  const [deployer] = await ethers.getSigners();
  console.log("Deploying with account:", deployer.address);

  const Microgrant = await ethers.getContractFactory("Microgrant");
  const microgrant = await Microgrant.deploy(); // no constructor args
  await microgrant.deployed();

  console.log("Microgrant deployed to:", microgrant.address);
}

main()
  .then(() => process.exit(0))
  .catch((err) => {
    console.error(err);
    process.exit(1);
  });
```

Run these commands:

```
bash Copy code

# compile
npx hardhat compile

# deploy to Primordial (make sure .env has PRIVATE_KEY and BDAG_RPC_URL)
npx hardhat run scripts/deploy.js --network primordial
```

After deploy, the script will print the contract address. Save it for the frontend.

10 — Get test BDAG (Faucet) & verify

1. In MetaMask, switch to the BDAG network (use step in frontend to add network programmatically or add manually). BlockDAG docs show the network params (chain id 1043, RPC above).
2. Visit the Primordial faucet: <https://primordial.bdagscan.com/faucet>, paste your wallet address, request tokens.

3. Use explorer to verify transactions: <https://primordial.bdagscan.com> and search for tx hash or address.

11 — Frontend (very small) — frontend/index.html + frontend/app.js

Create frontend/index.html:

```
html Copy code

<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>CodeQueens Microgrants</title>
  <script src="https://cdn.jsdelivr.net/npm/ethers@6/dist/ethers.min.js"></script>
</head>
<body>
  <h1>CodeQueens Microgrants (BDAG)</h1>

  <button id="btnAddNetwork">Add BDAG Network</button>
  <button id="btnConnect">Connect MetaMask</button>
  <div>Account: <span id="acct">--</span></div>
  <div>Balance: <span id="bal">--</span></div>


  <h3>Allocate (owner only)</h3>
  <input id="allocAddr" placeholder="recipient address"/>
  <input id="allocAmount" placeholder="amount (BDAG)"/>
  <button id="btnAlloc">Allocate</button>
```

```
  <h3>Payout (owner only)</h3>
  <input id="payAddr" placeholder="recipient address"/>
  <button id="btnPayout">Payout</button>

  <script src="./app.js"></script>
</body>
</html>
```

Create `frontend/app.js` and paste your contract address & ABI (replace placeholders):


js

 Copy code

```
const CONTRACT_ADDRESS = "0xYOUR_DEPLOYED_CONTRACT_ADDRESS";
const ABI = [ /* paste ABI from artifacts/contracts/Microgrant.sol/Microgrant.json */ ]

const BDAG_PARAMS = {
  chainId: "0x413",
  chainName: "BlockDag",
  nativeCurrency: { name: "BlockDAG", symbol: "BDAG", decimals: 18 },
  rpcUrls: ["https://rpc.primordial.bdagscan.com"],
  blockExplorerUrls: ["https://primordial.bdagscan.com"]
};

async function addNetwork() {
  if (!window.ethereum) return alert("Install MetaMask");
  try {
    await window.ethereum.request({
      method: "wallet_addEthereumChain",
      params: [BDAG_PARAMS],
    });
    alert("Network added / ready");
  } catch (err) {
    console.error(err);
    alert(err.message || "Failed to add network");
  }
}
```



```

let provider, signer, contract;
async function connect() {
  await window.ethereum.request({ method: "eth_requestAccounts" });
  provider = new ethers.BrowserProvider(window.ethereum);
  signer = await provider.getSigner();
  const addr = await signer.getAddress();
  document.getElementById("acct").textContent = addr;
  const bal = await provider.getBalance(addr);
  document.getElementById("bal").textContent = ethers.formatEther(bal);

  contract = new ethers.Contract(CONTRACT_ADDRESS, ABI, signer);
}

async function allocate() {
  const addr = document.getElementById("allocAddr").value;
  const amount = document.getElementById("allocAmount").value;
  if (!addr || !amount) return alert("addr + amount required");
  // convert to wei
  const amountWei = ethers.parseEther(amount);
  const tx = await contract.allocate(addr, amountWei);
  alert("Allocation tx submitted: " + tx.hash);
  await tx.wait();
  alert("Allocation confirmed.");
}

```

```

async function payout() {
  const addr = document.getElementById("payAddr").value;
  if (!addr) return alert("recipient required");
  const tx = await contract.payout(addr);
  alert("Payout tx: " + tx.hash);
  await tx.wait();
  alert("Payout confirmed.");
}


document.getElementById("btnAddNetwork").onclick = addNetwork;
document.getElementById("btnConnect").onclick = connect;
document.getElementById("btnAlloc").onclick = allocate;
document.getElementById("btnPayout").onclick = payout;

```

Notes: paste the ABI found at `artifacts/contracts/Microgrant.sol/Microgrant.json` produced by Hardhat after compile. Or use `npm run hardhat compile` then copy "abi" array.

12 — Full command sequence (copy/paste)

bash

 Copy code

```
# 0. prerequisites: installed Node, VS Code, MetaMask

# 1. create project
mkdir codequeens-microgrants && cd codequeens-microgrants
npm init -y

# 2. install packages
npm install --save-dev hardhat @nomicfoundation/hardhat-toolbox
npm install @openzeppelin/contracts dotenv

# 3. init hardhat (choose JS project)
npx hardhat

# 4. create files as instructed above (contracts/, scripts/, frontend/, .env)

# 5. compile
npx hardhat compile

# 6. put PRIVATE_KEY and BDAG_RPC_URL into .env (use the exported private key from a test account)

# 7. deploy to Primordial
npx hardhat run scripts/deploy.js --network primordial

# 8. open frontend (serve statically e.g. VS Code Live Server or python)
# example using simple python server in frontend folder
cd frontend
python -m http.server 5500
# browse http://localhost:5500
```

13 — Additional tips & checklist for demo day

- Use only test accounts and explicitly say so. Never use exported keys in public demos for real accounts. support.metamask.io
- If deployment fails, check that the deployer address has BDAG tokens from the faucet. primordial.bdagscan.com
- Keep a screenshot or video of the happy path (connect → show balance → deploy or allocate → payout tx on explorer) as fallback.

- Map flows to judging criteria: transparency (all txs on explorer), traceability (allocations logged), low-cost transfers (test small BDAG values), and ease-of-use (owner UI).

14 — Useful commands & extras

- Install TypeScript tooling (if you want TS scripts):

```
bash

npm install --save-dev typescript ts-node @types/node
npx tsc --init
```

[Copy code](#)

- Add helpful npm scripts to `package.json`:

```
json

"scripts": {
  "compile": "npx hardhat compile",
  "deploy:primordial": "npx hardhat run scripts/deploy.js --network primordial",
  "start-frontend": "cd frontend && python -m http.server 5500"
}
```

[Copy code](#)

15 — Why these packages

- Hardhat + `@nomicfoundation/hardhat-toolbox` is the current recommended setup for compiling, testing, and deploying EVM contracts with built-in ethers testing tooling. [hardhat.org](#) [npm](#)
- OpenZeppelin is the standard, audited library for secure contract primitives like `Ownable`. [docs.openzeppelin.com](#)

16 — Final security reminder (must-read)

- **Never** commit `.env` or private keys. Use a test-only key. MetaMask private key export is irreversible — do it only on throwaway/test accounts and store the key in a safe local place while developing. If you ever suspect a key is leaked, move funds immediately (for mainnet) and **do not** reuse the key.

[support.metamask.io](#)

.gitignore file

Node modules

node_modules/
npm-debug.log*
yarn-debug.log*
yarn-error.log*

Hardhat & build artifacts
artifacts/
cache/
coverage/
coverage.json
typechain/
typechain-types/

Environment & secrets
.env
.env.local
.env.*.local

Logs
logs
*.log
*.csv
*.dat
*.out
*.pid
*.seed

VSCode / IDE
.vscode/
.idea/
*.swp
*.swo

OS-specific
.DS_Store
Thumbs.db

Frontend build outputs (if you add a bundler later like Vite/Webpack/React)
dist/
build/

Misc
*.tgz
*.zip
*.tar.gz

Link to Chat GPT conversation

<https://chatgpt.com/share/68aec031-549c-8006-8982-8337d9f282a8>