

# GridSmart Energy MVP - Setup & Deployment Guide

## Quick Start (MVP Setup)

This MVP includes the **bare minimum functionality** to demonstrate your GridSmart Energy concept at the hackathon.

### What's Included (MVP Features)

#### Smart Contracts

- Basic transformer load monitoring
- Simple incentive commitment system
- Peer-to-peer energy trading marketplace

#### Mock AI Predictions

- Realistic load forecasting simulation
- South African energy consumption patterns

#### BlockDAG Integration

- Testnet deployment ready
- Web3 wallet connection

### What's Abstract (Documentation Only)

#### Production Features (skeleton + docs):

- Real SCADA system integration
- Advanced ML model training
- Production oracle services
- Complex grid analytics
- Enterprise security features

---

## Installation & Setup

### 1. Prerequisites

```
bash

# Install Node.js 18+
# Install Git
# Get a crypto wallet (MetaMask/Trust Wallet)
```

## 2. Project Setup

```
bash

# Clone/setup project
mkdir gridsmart-energy-mvp
cd gridsmart-energy-mvp

# Initialize npm project
npm init -y

# Install dependencies
npm install --save-dev hardhat @nomiclabs/hardhat-waffle @nomiclabs/hardhat-ethers
npm install ethers dotenv @walletconnect/web3-provider
```

## 3. Environment Configuration

Create `.env` file:

```
env

# Get your private key from MetaMask/wallet
PRIVATE_KEY=your_wallet_private_key_here

# BlockDAG testnet RPC
BLOCKDAG_RPC_URL=https://rpc-test.blockdagnetwork.io

# Optional: BlockDAG explorer API key
BLOCKDAG_API_KEY=dummy_for_now
```

## 4. Get Testnet Tokens

- Visit: <https://awakening.bdagscan.com/faucet>
- Connect your wallet
- Request BDAG testnet tokens
- Confirm you received tokens



## Deployment Process

### 1. Compile Contracts

```
bash

npx hardhat compile
```

### 2. Deploy to BlockDAG Testnet

```
bash
```

```
npx hardhat run scripts/deploy.js --network blockdag_testnet
```

### 3. Verify Deployment

After deployment, you'll see:

🎉 Deployment Complete!

TransformerLoadManager: <https://testnet.bdagscan.com/address/0x...>

IncentiveManager: <https://testnet.bdagscan.com/address/0x...>

P2PEnergyTrading: <https://testnet.bdagscan.com/address/0x...>

### 4. Update Frontend

Copy contract addresses from `deployment-info.json` to your frontend:

```
javascript
```

```
// Update in web3Integration.js
```

```
this.contractAddresses = {
```

```
  loadManager: "0x...", // From deployment-info.json
```

```
  incentiveManager: "0x...",
```

```
  p2pTrading: "0x..."
```

```
};
```

---

## 🤖 Demo Script (5-Minute Hackathon Demo)

### Setup (Before Demo)

1. Deploy contracts ✅
2. Fund incentive pool with 10 BDAG ✅
3. Create sample energy listing ✅
4. Have 2-3 wallet addresses ready for demo

### Demo Flow

#### Minute 1: Problem Statement

- Show Eskom stats (R2.8B transformer failures)
- Explain load shedding costs (R899M/day)

#### Minute 2: Show Current Load

- Display transformer at 55% load
- AI predicts 72% load next hour
- Risk level: "MEDIUM" → "HIGH"

### Minute 3: Incentive System

- User commits to reduce 5 kWh
- System calculates reward (2x rate = 200 wei/kWh)
- Mock verification shows successful reduction
- User claims 1000 wei reward

### Minute 4: P2P Trading

- Solar user lists 5 kWh at 0.001 BDAG/kWh
- Regular user buys 3 kWh
- Transaction confirmed on BlockDAG
- Shows remaining 2 kWh still available

### Minute 5: Results & Impact

- Transformer load reduced to 48%
- Critical threshold avoided
- Users earned rewards
- P2P market active

---

## Architecture Overview

### MVP Smart Contracts

#### 1. TransformerLoadManager.sol

```
solidity

// Core Functions (Working)
- updateCurrentLoad(uint256) // Oracle updates
- submitPrediction(uint256, uint256) // AI predictions
- getCurrentLoad() // Frontend reads
- isCriticalLoad() // Alert system

// Events (Working)
- LoadUpdated
- PredictionMade
- CriticalLoadWarning
```

## 2. IncentiveManager.sol

solidity

```
// Core Functions (Working)
- commitToReduction(uint256) // User commits
- claimRewards() // User gets paid
- fundRewardsPool() // Eskom funds system
- verifyAndReward() // Mock verification

// User tracking (Working)
- pendingRewards mapping
- userStats mapping
```

## 3. P2PEnergyTrading.sol

solidity

```
// Core Functions (Working)
- listEnergy() // Sellers list kWh
- buyEnergy() // Buyers purchase
- getActiveListings() // Browse market
- cancelListing() // Seller cancels

// Market data (Working)
- EnergyListing struct
- Trade history
- User trading stats
```

## Mock Services (Working)

### MockDataService.js

- Generates realistic SA load patterns
- Morning peak: 6-9 AM (75-80% load)
- Evening peak: 5-8 PM (80-88% load)
- Weekend adjustments
- Temperature effects on demand

### Web3Integration.js

- WalletConnect for mobile wallets
  - MetaMask browser support
  - Real-time event listening
  - Error handling & user feedback
-

## What Would Be Built for Production

### 1. Real Data Integration

javascript

*// services/scadaIntegration.js (ABSTRACT)*

```
class SCADAIntegration {  
  // Connect to Eskom's SCADA systems  
  // Pull real transformer data every 30 seconds  
  // Handle data validation & cleaning  
  // Push to blockchain via secure oracle  
}
```

*// services/weatherAPI.js (ABSTRACT)*

```
class WeatherIntegration {  
  // Connect to SAWS weather API  
  // Temperature affects energy demand  
  // Integrate with load prediction model  
}
```

### 2. Advanced ML Pipeline

python

*# models/transformerPredictor.py (ABSTRACT)*

**class** ProductionMLModel:

"""

Production ML would include:

- XGBoost ensemble models
- LSTM for time series
- Feature engineering (20+ variables)
- Model retraining pipeline
- A/B testing framework
- Uncertainty quantification

"""

**def** train\_model(self, historical\_data):

*# Process 2+ years of transformer data*

*# Weather, economic indicators, events*

*# Time-of-day, seasonal patterns*

**pass**

**def** predict\_load(self, current\_state):

*# Ensemble prediction from multiple models*

*# Return prediction + confidence intervals*

*# Risk assessment & recommendations*

**pass**

### 3. Enterprise Security

solidity

*// contracts/EnterpriseLoadManager.sol (ABSTRACT)*

**contract** EnterpriseLoadManager {

*// Multi-signature oracle validation*

*// Role-based access control (Eskom operators)*

*// Circuit breakers for emergency stops*

*// Audit trail for regulatory compliance*

*// Integration with national grid protocols*

}

### 4. Advanced Oracle Network

```
javascript
```

```
// services/oracleNetwork.js (ABSTRACT)
class DecentralizedOracle {
  // Multiple data source validation
  // Consensus mechanism for data accuracy
  // Cryptographic proofs for data integrity
  // Fallback systems for network failures
  // Integration with Chainlink oracles
}
```

## 5. Grid Analytics Dashboard

```
javascript
```

```
// components/GridAnalyticsDashboard.js (ABSTRACT)
class GridAnalytics {
  // Real-time national grid visualization
  // Predictive load maps by region
  // Economic impact calculations
  // Regulatory reporting automation
  // Integration with Eskom systems
}
```

## Testing Your MVP

### Manual Testing Checklist

#### Smart Contracts

```
bash
```

```
# Test transformer load updates
npx hardhat console --network blockdag_testnet
> const contract = await ethers.getContractAt("TransformerLoadManager", "0x...")
> await contract.updateCurrentLoad(75)
> await contract.getCurrentLoad() // Should return 75

# Test incentive system
> const incentives = await ethers.getContractAt("IncentiveManager", "0x...")
> await incentives.commitToReduction(5)
> await incentives.getPendingRewards(yourAddress)
```

### Frontend Integration



```
javascript
```

```
// Test wallet connection
const result = await web3Integration.connectWallet();
console.log("Connected:", result.success);

// Test load monitoring
const load = await web3Integration.getCurrentLoad();
console.log("Current load:", load.currentLoad + "%");

// Test incentive commitment
const commitment = await web3Integration.commitToReduction(5);
console.log("Commitment TX:", commitment.txHash);
```

## Demo Day Preparation

### 1. Pre-Demo Setup (30 mins before)

- Deploy fresh contracts to testnet
- Fund incentive pool with demo tokens
- Create 2-3 sample energy listings
- Test wallet connections on demo device
- Prepare backup slides in case of network issues

### 2. Live Demo Backup Plan

If blockchain demo fails:

- Show local hardhat network version
- Use pre-recorded transaction videos
- Focus on smart contract code walkthrough
- Emphasize production architecture plans

### 3. Questions to Expect

- "How do you prevent gaming the incentive system?"
- "What's the economic model for P2P pricing?"
- "How do you handle transformer ownership/regulation?"
- "What's the AI model accuracy in production?"
- "How does this integrate with Eskom's existing systems?"

Feature	MVP Status	Production Plan
Transformer Monitoring	✔️ Mock data simulation	🔧 Real SCADA integration
Load Prediction	✔️ Rule-based ML mock	🔧 XGBoost ensemble model
Incentive System	✔️ Basic commitment/reward	🔧 Smart verification system
P2P Trading	✔️ Simple marketplace	🔧 Advanced matching engine
Oracle Service	✔️ Manual admin updates	🔧 Decentralized oracle network
Security	✔️ Basic access control	🔧 Enterprise-grade security
Scalability	✔️ Single transformer	🔧 National grid scale
Regulatory	✔️ Demo compliance	🔧 NERSA/Eskom integration

## Known MVP Limitations

### What Works for Demo

- Basic smart contracts deployed and functional
- Simple incentive mechanics
- P2P energy marketplace
- Mock AI predictions with realistic patterns
- Web3 wallet integration

### What's Simplified for MVP

- **Verification System:** Currently admin-approved, production needs IoT sensors
- **Oracle Data:** Manual updates vs. real-time SCADA feeds
- **AI Model:** Rule-based vs. trained ML models
- **Security:** Basic vs. enterprise-grade access controls
- **Scale:** Single transformer vs. grid-wide deployment

### Production Development Timeline

- **Phase 1 (3 months):** Real data integration, basic ML training
- **Phase 2 (6 months):** Advanced AI models, pilot with Eskom
- **Phase 3 (12 months):** Full grid deployment, regulatory approval

## Support & Resources

### BlockDAG Resources

- **Testnet Faucet:** <https://awakening.bdagscan.com/faucet>
- **Explorer:** <https://testnet.bdagscan.com>
- **Documentation:** <https://docs.blockdagnetwork.io/>
- **IDE:** <https://ide.primordial.bdagscan.com/>

## South African Context

- **Eskom Load Patterns:** Peak 6-9 AM, 5-8 PM
- **Transformer Costs:** R2.8 billion per major failure
- **Load Shedding Impact:** R899 million per day at Stage 6
- **Grid Stats:** 6,000 power transformers + 400,000 distribution transformers

## Demo Day Tips

- **Emphasize Impact:** Focus on the R2.8B transformer failure cost
  - **Show Real Data:** Use authentic SA load patterns
  - **Technical Depth:** Be ready to dive into smart contract code
  - **Production Vision:** Clearly explain MVP vs. production differences
  - **Regulatory Awareness:** Acknowledge Eskom/NERSA integration needs
- 

## You're Ready!

Your MVP is now deployed and ready for the hackathon demo. The system demonstrates:

1. **Real Problem:** Transformer overload prevention
2. **Blockchain Solution:** Smart contracts on BlockDAG
3. **Economic Incentives:** Pay users to reduce consumption
4. **Market Innovation:** P2P energy trading
5. **AI Integration:** Predictive load forecasting

**Remember:** This is an MVP to prove the concept. Production would require significant additional development, but you've built a solid foundation that shows the potential impact and technical feasibility.

Good luck with your hackathon! 