Activity 2 — Hardhat End-to-End (Local): Deploy, Transact, Analyze

Duration: 90–120 minutes

Outcome: A fully working local project using Hardhat + ethers (v6), OpenZeppelin ERC-20, deploy + interact + analyze transactions (including EIP-1559 fee fields and event decoding).

0) Pre-flight (5 min)

All commands below run **inside the student's Ubuntu (latest LTS) VM**.

```
node -v      # Expect v20.x
npm -v       # Expect v10.x
```

Create a clean workspace and open the editor:

```
mkdir -p ~/campuscredit && cd ~/campuscredit
code .
```

1) Initialize the Hardhat project (10 min)

```
npm init -y
npm install --save-dev hardhat @nomicfoundation/hardhat-toolbox dotenv
npx hardhat init   # Choose: "Create a JavaScript project"
npm install @openzeppelin/contracts
```

Create a minimal `.gitignore` (keep repos clean):

```
cat > .gitignore << 'EOF'
node_modules/
.env
coverage/
cache/
artifacts/
EOF
```

Create an empty `.env` (kept for later weeks; not used today):

```
echo 'DUMMY=1' > .env
```

2) Configure Hardhat for EIP-1559 (5 min)

We want meaningful **base fee** and **priority tip** on the local chain so students can analyze fees.

Edit **`hardhat.config.js`**:

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

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.24",
  networks: {
    hardhat: {
      // Enable EIP-1559 fields locally (set a nonzero base fee)
      initialBaseFeePerGas: 1_000_000_000, // 1 gwei
      // You can also add chainId: 31337 (default) if you plan to
connect MetaMask later
      chainId: 31337,
    },
    localhost: {
      url: "http://127.0.0.1:8545",
      chainId: 31337,
```

```
    },
  },
};
```

3) Write the ERC-20 token (10 min)

Create **`contracts/CampusCredit.sol`**:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

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

contract CampusCredit is ERC20 {
    constructor(uint256 initialSupply) ERC20("CampusCredit", "CAMP") {
        _mint(msg.sender, initialSupply); // Mint full supply to
deployer
    }
}
```

Compile:

```
npx hardhat compile
```

✅ **Checkpoint:** Compilation succeeds.

4) Start a local Hardhat node (5 min)

In **Terminal A**:

```
npx hardhat node
```

You'll see **20 funded test accounts** with private keys (local only). Leave this running.

5) Deploy to the local network (10 min)

Create **`scripts/deploy.js`**:

```
const { ethers } = require("hardhat");

async function main() {
  const initialSupply = ethers.parseUnits("1000000", 18); // 1,000,000
CAMP
  const Token = await ethers.getContractFactory("CampusCredit");
  const token = await Token.deploy(initialSupply);
  await token.waitForDeployment();

  const address = await token.getAddress();
  const [deployer] = await ethers.getSigners();

  console.log("CampusCredit deployed to:", address);
  console.log("Deployer:", deployer.address);
  const bal = await token.balanceOf(deployer.address);
  console.log("Deployer CAMP balance:", ethers.formatUnits(bal, 18));
}

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

In **Terminal B** (same folder), deploy to the node you just started:

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

Copy the printed **contract address** (you'll paste it below).

✅ **Checkpoint:** You see the token address and deployer's CAMP balance = 1,000,000.

6) Interact: transfers, approval, different tips (15–20 min)

Create **`scripts/interact.js`** (paste your token address):

```
const { ethers } = require("hardhat");

// Paste the address printed by deploy.js
const TOKEN = "<PASTE_DEPLOYED_ADDRESS>";

async function main() {
  if (!TOKEN) throw new Error("Set TOKEN address in
scripts/interact.js");

  const [deployer, acct2] = await ethers.getSigners();
  const token = await ethers.getContractAt("CampusCredit", TOKEN,
deployer);

  // Helper to print balances
  async function balances(label) {
    const b1 = await token.balanceOf(deployer.address);
    const b2 = await token.balanceOf(acct2.address);
    console.log(`${label} | Deployer: ${ethers.formatUnits(b1, 18)}
CAMP | Acct2: ${ethers.formatUnits(b2, 18)} CAMP`);
  }

  await balances("Before");

  // Transfer #1 with lower priority tip
  const tx1 = await token.transfer(acct2.address,
    ethers.parseUnits("100", 18),
    { maxPriorityFeePerGas: 1_000_000_000n, maxFeePerGas:
20_000_000_000n } // 1 gwei tip
  );
  console.log("Tx1 hash:", tx1.hash);
  const rcpt1 = await tx1.wait();
  console.log("Tx1 mined in block:", rcpt1.blockNumber);
```

```
  // Transfer #2 with higher priority tip
  const tx2 = await token.transfer(
    acct2.address,
    ethers.parseUnits("50", 18),
    { maxPriorityFeePerGas: 3_000_000_000n, maxFeePerGas:
22_000_000_000n } // 3 gwei tip
  );
  console.log("Tx2 hash:", tx2.hash);
  const rcpt2 = await tx2.wait();
  console.log("Tx2 mined in block:", rcpt2.blockNumber);

  // Approval: allow acct2 to spend 25 CAMP
  const tx3 = await token.approve(
    acct2.address,
    ethers.parseUnits("25", 18),
    { maxPriorityFeePerGas: 2_000_000_000n, maxFeePerGas:
21_000_000_000n } // 2 gwei tip
  );
  console.log("Tx3 hash:", tx3.hash);
  const rcpt3 = await tx3.wait();
  console.log("Tx3 mined in block:", rcpt3.blockNumber);

  await balances("After");

  console.log("HASHES:", JSON.stringify({ tx1: tx1.hash, tx2:
tx2.hash, tx3: tx3.hash }, null, 2));
}

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

Run:

```
npx hardhat run scripts/interact.js --network localhost
```

Copy the **3 tx hashes** printed (you'll use them in the analysis script).

✅ **Checkpoint:** You see “Before/After” balances and 3 tx hashes. You also see which blocks they mined in (useful when comparing tips).

7) Analyze: receipts, blocks, EIP-1559 fields, event decoding (15–20 min)

Create **`scripts/analyze.js`**:

```
const { ethers } = require("hardhat");

// Optional: for context printing only
const TOKEN = "<PASTE_DEPLOYED_ADDRESS_FROM_DEPLOY>";

const HASHES = {
  tx1: "<PASTE_FROM_INTERACT_OUTPUT>",
  tx2: "<PASTE_FROM_INTERACT_OUTPUT>",
  tx3: "<PASTE_FROM_INTERACT_OUTPUT>",
};

const iface = new ethers.Interface([
  "event Transfer(address indexed from, address indexed to, uint256
value)",
  "event Approval(address indexed owner, address indexed spender,
uint256 value)",
]);

async function analyze(hash) {
  const tx = await ethers.provider.getTransaction(hash);
  const rcpt = await ethers.provider.getTransactionReceipt(hash);
  const block = await ethers.provider.getBlock(rcpt.blockNumber);

  const baseFee = block.baseFeePerGas ?? 0n;
  const gasUsed = rcpt.gasUsed ?? 0n;
  const effective = rcpt.effectiveGasPrice ?? tx.gasPrice ?? 0n;
  const totalFee = gasUsed * effective;
```

```
  console.log("\n=== Analysis for", hash, "===");
  console.log("Status:", rcpt.status === 1 ? "Success" : "Fail");
  console.log("Block number:", rcpt.blockNumber);
  console.log("Timestamp (UTC):", new Date(Number(block.timestamp) *
1000).toISOString());
  console.log("From:", tx.from);
  console.log("To:", tx.to);
  console.log("Nonce:", tx.nonce);
  console.log("Gas limit:", tx.gasLimit?.toString());
  console.log("Gas used:", gasUsed.toString());
  console.log("Base fee per gas:", baseFee.toString());
  console.log("Max fee per gas:", (tx.maxFeePerGas ?? 0n).toString());
  console.log("Max priority fee per gas:", (tx.maxPriorityFeePerGas ??
0n).toString());
  console.log("Effective gas price:", effective.toString());
  console.log("Total fee (wei):", totalFee.toString());

  // Decode Transfer/Approval events
  for (const log of rcpt.logs) {
    try {
      const parsed = iface.parseLog({ topics: log.topics, data:
log.data });
      // Pretty print decoded values
      if (parsed.name === "Transfer" || parsed.name === "Approval") {
        const args = parsed.args;
        const val = args.value ? ethers.formatUnits(args.value, 18) :
"";
        console.log(`Event: ${parsed.name}`, {
          from: args.from ?? args.owner,
          to: args.to ?? args.spender,
          valueRaw: args.value?.toString(),
          valueHuman: val,
        });
      } else {
        console.log("Event:", parsed.name, parsed.args);
      }
    } catch (_) { /* Ignore logs from other contracts */ }
  }
}
```

```
async function main() {
  console.log("Token (context):", TOKEN);
  await analyze(HASHES.tx1);
  await analyze(HASHES.tx2);
  await analyze(HASHES.tx3);
}

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

Run:

```
npx hardhat run scripts/analyze.js --network localhost
```

Take screenshots of the console output. Students will use these in `report.md`.

✅ **Checkpoint:** You see all fields (nonce, gas, fees, effective gas price, total fee) and **decoded events** (`Transfer`, `Approval`) with raw and human-readable amounts.

8) (Optional, if time) Hardhat console & MetaMask (10–15 min)

A) Hardhat console

```
npx hardhat console --network localhost
```

Inside the console:

```
const [deployer, acct2] = await ethers.getSigners();
const token = await ethers.getContractAt("CampusCredit",
"<TOKEN_ADDRESS>", deployer);
( await token.balanceOf(deployer.address) ).toString();
```

```
( await token.balanceOf(acct2.address) ).toString();
await token.transfer(acct2.address, ethers.parseUnits("1", 18));
```

B) MetaMask connection

- Network name: **Hardhat Local**
- RPC URL: http://127.0.0.1:8545
- Chain ID: **31337**
- Currency symbol: **ETH**

Import one **local** account (copy a private key from Terminal A's node output). **Use only for local!** Never reuse anywhere else.

You can then send a small transfer via MetaMask and see it appear in **Terminal A** logs.

9) (Optional) Minimal unit tests (10 min)

Create **`test/CampusCredit.test.js`**:

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("CampusCredit", function () {
  it("mints full initial supply to deployer and transfers correctly",
async function () {
    const [deployer, acct2] = await ethers.getSigners();
    const initialSupply = ethers.parseUnits("1000000", 18);
    const Token = await ethers.getContractFactory("CampusCredit");
    const token = await Token.deploy(initialSupply);
    await token.waitForDeployment();

    expect(await
token.balanceOf(deployer.address)).to.equal(initialSupply);

    await token.transfer(acct2.address, ethers.parseUnits("100", 18));
    expect(await
```

```
token.balanceOf(acct2.address)).to.equal(ethers.parseUnits("100",
18));
  });

  it("approves and updates allowance", async function () {
    const [deployer, spender] = await ethers.getSigners();
    const Token = await ethers.getContractFactory("CampusCredit");
    const token = await Token.deploy(ethers.parseUnits("1000", 18));
    await token.waitForDeployment();

    await token.approve(spender.address, ethers.parseUnits("25", 18));
    expect(await token.allowance(deployer.address, spender.address))
      .to.equal(ethers.parseUnits("25", 18));
  });
});
```

Run:

```
npx hardhat test
```

✅ **Checkpoint:** All tests pass (2 passing).

10) Commit to GitHub (5 min)

Use the **Assignment 2 repo** naming format later; for class, you can push to a practice repo.

```
git init
git add .
git commit -m "Class activity: Hardhat deploy + interact + analyze"
# git remote add origin https://github.com/<you>/campuscredit-hardhat-demo.git
# git branch -M main && git push -u origin main
```

Troubleshooting Guide (keep on screen)

- **`TypeError: ... parseUnits is not a function`**
  - You're likely mixing ethers v5 snippets with v6. This activity uses **ethers v6** via `@nomicfoundation/hardhat-toolbox`. Use `ethers.parseUnits` / `ethers.formatUnits` and `await token.waitForDeployment()`.
- **Deployment connects to the wrong network**
  - Include `--network localhost` when the node is running; otherwise Hardhat uses the in-process network (fine, but hashes won't match Terminal A).
- **No base fee / EIP-1559 fields show as null**
  - Ensure `initialBaseFeePerGas` is set in `hardhat.config.js` under the `hardhat` network. Restart `npx hardhat node` after changes.
- **`CALL_EXCEPTION / insufficient funds`**
  - You're not using a funded local account. Ensure you're using signers from `ethers.getSigners()` on the **localhost** network.
- **`invalid BigNumber string`**
  - Always pass string amounts to `parseUnits`, e.g., `ethers.parseUnits("100", 18)`.
- **Scripts can't find the contract**
  - Double-check you pasted the correct token address from `deploy.js` into `interact.js` and `analyze.js`.

What students should have at the end

- A working Hardhat project with:
  - `contracts/CampusCredit.sol`
  - `scripts/deploy.js`, `scripts/interact.js`, `scripts/analyze.js`
  - `hardhat.config.js`, `.gitignore`, `.env` (empty)
- **Local node** logs in Terminal A.
- **Deploy** output (token address, balances).
- **Interact** output (3 tx hashes; before/after balances).
- **Analyze** output with **nonce, gas limit, gas used, base fee, max fee, max priority fee, effective gas price, total fee**, and **decoded Transfer/Approval events**.
- (Optional) **Passing tests** and/or **MetaMask connected** to localhost.