ABYA University Decentralized Learning Platform Integration

This documentation provides a detailed step-by-step procedure for integrating various modules into the Learning Management System (LMS) of ABYA University using the Alphabill blockchain and other decentralized technologies.

A. Wallet

Must-Have Features for the MVP:

1. *User Authentication and Wallet Connection:* Allow users to connect their Metamask wallet to the platform.
2. *Balance Display:* Show the user's balance in their connected wallet.
3. *Transaction Management:* Enable users to send and receive tokens.
4. *Security Education:* Provide educational resources on securing their wallet and recognizing phishing attempts.
5. *Compatibility:* Ensure compatibility with the Alphabill blockchain's EVM partition.

Step-by-Step Integration Procedure:

Step 1: Install Metamask

- Ensure that users install the Metamask browser extension. Metamask Installation Guide

Step 2: Connect Metamask to the Application

- Create a button in your LMS that users can click to connect their Metamask wallet.
- Use Web3.js or Ethers.js to handle the connection process.

`// Example with Ethers.js import from "ethers";

async function connectWallet() { if (window.ethereum) { try { await window.ethereum.request({ method: "eth_requestAccounts" }); const provider = new ethers.providers.Web3Provider(window.ethereum); const signer = provider.getSigner(); console.log("Account:", await signer.getAddress()); } catch (error) { console.error("User denied account access"); } } else { console.log("Metamask not found"); } } `

Step 3: Display User's Balance

- Fetch and display the user's balance using Web3.js or Ethers.js.

```
async function getBalance() { const provider = new
ethers.providers.Web3Provider(window.ethereum); const signer =
provider.getSigner(); const balance = await signer.getBalance();
console.log("Balance:", ethers.utils.formatEther(balance)); }
```

Step 4: Transaction Management

- Allow users to send transactions from their Metamask wallet.

```
async function sendTransaction() { const provider = new
ethers.providers.Web3Provider(window.ethereum); const signer =
provider.getSigner(); const tx = await signer.sendTransaction({ to:
"recipient_address_here", value: ethers.utils.parseEther("0.01") });
console.log("Transaction:", tx); }
```

Step 5: Security Education

- Provide resources and guidelines for users to secure their Metamask wallet. Metamask Security Tips

Step 6: Ensure Compatibility with Alphabill EVM Partition

- Configure Metamask to connect to the Alphabill EVM partition.

`const alphabillNetwork = { chainId: '0x... (your chain id)', chainName: 'Alphabill Testnet', rpcUrls: ['https://... (your RPC URL)'], nativeCurrency: { name: 'Alphabill', symbol: 'AB', decimals: 18 }, blockExplorerUrls: ['https://... (your block explorer URL)'] };

async function addAlphabillNetwork() { try { await window.ethereum.request({ method: 'wallet_addEthereumChain', params: [alphabillNetwork] }); } catch (error) { console.error("Failed to add network", error); } } `

resources

- Metmask Documentation
- Ethers.js Documentation

B. RPC provider

Must-Have Features for the MVP:

1. *Reliable Network Connection:* Ensure a stable connection to the Alphabill blockchain.
2. *Scalability:* Handle multiple requests efficiently.
3. *Real-Time Notifications:* Receive real-time updates and notifications.
4. *Enhanced APIs:* Access advanced APIs for improved functionality.

Step-by-Step Integration Procedure:

Step 1: Sign Up and Get API Key

- Sign up for Alchemy and obtain an API key.
- Alchemy Signup

Step 2: Configure Alchemy as RPC Provider

- Integrate Alchemy as the RPC provider in your application.

`import from "@ethersproject/providers";

const alchemyApiKey = "your-alchemy-api-key"; const provider = new AlchemyProvider("homestead", alchemyApiKey); `

Step 3: Implement Enhanced APIs

- Utilize Alchemy's enhanced APIs for better functionality.

`async function getBlockNumber() { const blockNumber = await provider.getBlockNumber(); console.log("Current Block Number:", blockNumber); }

async function getTransactionReceipt(txHash) { const receipt = await provider.getTransactionReceipt(txHash); console.log("Transaction Receipt:", receipt); } `

Step 4: Real-Time Notifications

- Implement real-time notifications using Alchemy's WebSocket provider.

`import from "@ethersproject/providers";

const wsProvider = new WebSocketProvider(`wss://eth-mainnet.ws.alchemyapi.io/v2/${alchemyApiKey}`);

wsProvider.on("block", (blockNumber) => { console.log("New Block:", blockNumber); }); `

Step 5: Monitor Network Activity

- Use Alchemy's dashboard to monitor network activity and performance. Alchemy Dashboard

Resources:

- Alchemy Documentation
- Ethers.js Provider Documentation

C. Blockchain Platform

Must-Have Features for the MVP:

1. Smart Contract Deployment: Ability to deploy and manage smart contracts on the Alphabill blockchain.
2. Scalability: Handle high transaction throughput with low latency.
3. Token Programmability: Create and manage custom tokens for the learning ecosystem.
4. Cost-Effective Transactions: Ensure low transaction fees to encourage platform usage.
5. EVM Compatibility: Seamless integration with existing EVM-based tools and libraries.

Step-by-Step Integration Procedure:

Step 1: Setup Development Environment

- Ensure you have Node.js and Hardhat installed. `npm install --save-dev hardhat`

Step 2: Initialize Hardhat Project

- Create a new Hardhat project and setup your configuration.

```
npx hardhat
```

Step 3: Configure Alphabill Network

- Add Alphabill network configuration to hardhat.config.js.

```
module.exports = { solidity: "0.8.4", networks: { alphabill: { url:
"https://rpc.alphabill.io", accounts: [process.env.PRIVATE_KEY] } } };
```

Step 4: Develop Smart Contracts

- Write and compile your smart contracts using Solidity.

`// contracts/ABToken.sol pragma solidity ^0.8.4;

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

contract ABToken is ERC20 { constructor(uint256 initialSupply) ERC20("ABToken", "ABT") { _mint(msg.sender, initialSupply); } } `

Step 5: Deploy Smart Contracts

- Deploy your smart contracts to the Alphabill blockchain.

`async function main() { const [deployer] = await ethers.getSigners();

```
console.log("Deploying contracts with the account:", deployer.address);

const Token = await ethers.getContractFactory("ABToken");
const token = await Token.deploy(1000000);

console.log("Token deployed to:", token.address);
```

}

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

Step 6: Interact with Deployed Contracts

- Use Ethers.js to interact with your deployed contracts.

`const contractAddress = "deployed-contract-address"; const abi = ["function totalSupply() public view returns (uint256)", "function balanceOf(address account) public view returns (uint256)", "function transfer(address recipient, uint256 amount) public returns (bool)"];

const provider = new ethers.providers.JsonRpcProvider("https://rpc.alphabill.io"); const wallet = new

ethers.Wallet(process.env.PRIVATE_KEY, provider); const contract = new ethers.Contract(contractAddress, abi, wallet);

async function getTotalSupply() { const totalSupply = await contract.totalSupply(); console.log("Total Supply:", totalSupply.toString()); }

async function getBalance(address) { const balance = await contract.balanceOf(address); console.log("Balance:", balance.toString()); }

async function transferTokens(recipient, amount) { const tx = await contract.transfer(recipient, amount); await tx.wait(); console.log("Transfer complete:", tx.hash); } `

Resources:

- Hardhat Documentation
- Alphabill Documentation
- OpenZeppelin Contracts

D. Auto Task

Must-Have Features for the MVP:

1. *Automated Contract Deployment:* Automatically deploy smart contracts to the Alphabill blockchain.
2. *Automated Testing:* Run automated tests on smart contracts to ensure reliability.
3. *Task Scheduling:* Schedule tasks for regular execution (e.g., data updates, contract interactions).

*Step-by-Step Integration Procedure:

Step 1: Setup Hardhat

- Ensure your Hardhat environment is set up as described in the Blockchain Platform section.

Step 2: Create Hardhat Tasks

- Define custom tasks in Hardhat to automate deployment and testing.

Example Task: Deploy Contract

- Create a script for contract deployment.

`task("deploy", "Deploys the smart contract") .setAction(async () => { const [deployer] = await ethers.getSigners(); console.log("Deploying contracts with the account:", deployer.address);

```
const Token = await ethers.getContractFactory("ABToken");
const token = await Token.deploy(1000000);

console.log("Token deployed to:", token.address);
```

});

module.exports = { solidity: "0.8.4", networks: { alphabill: { url: "https://rpc.alphabill.io", accounts: [process.env.PRIVATE_KEY] } } }; `

- Run the task.

`npx hardhat deploy --network alphabill` Example Task: Automated Testing

- Define a task to run tests.

`task("test", "Runs the smart contract tests") .setAction(async () => { await run("compile"); await run("test"); });

module.exports = { solidity: "0.8.4", networks: { alphabill: { url: "https://rpc.alphabill.io", accounts: [process.env.PRIVATE_KEY] } } }; `

- Run the test task.

```
npx hardhat test
```

Step 3: Automate Task Scheduling

- Use a task scheduler like cron to automate regular task execution.

Example: Scheduling with cron

- Create a script run-tasks.sh to execute your Hardhat tasks.

```
#!/bin/bash cd /path/to/your/project npx hardhat deploy --network alphabill npx hardhat test
```

- Schedule the script with `cron`.

`crontab -e` Add a line to schedule the script (e.g., to run every day at midnight).

```
0 0 * * * /path/to/run-tasks.sh
```

Resources:

- Hardhat Tasks Documentation
- Hardhat Network Documentation
- Crontab Guru

E. Indexing Platform

Must-Have Features for the MVP:

1. Data Aggregation: Aggregate and index data from the Alphabill blockchain.
2. Unified API Access: Provide a unified API for accessing blockchain data.
3. Real-Time Notifications: Set up real-time notifications for specific blockchain events.

Step-by-Step Integration Procedure:

Step 1: Setup Covalent

- Sign up for a Covalent API key at Covalent HQ.

Step 2: Retrieve Blockchain Data

- Use Covalent's unified API to retrieve blockchain data.

Example: Fetch Token Balances

- Create a script to fetch token balances.

`import requests

api_key = 'your_covalent_api_key' wallet_address = 'your_wallet_address' chain_id = 'alphabill_chain_id'

url = f'https://api.covalenthq.com/v1//address//balances_v2/?key='

response = requests.get(url) data = response.json()

for item in data['data']['items']: print(f"Token: {item['contract_name']}, Balance: {item['balance']}") `

Step 3: Integrate with LMS

- Create endpoints in your Django LMS to fetch and display blockchain data using Covalent's API.

Example: Django View for Token Balances

- Create a Django view to display token balances.

`import requests from django.shortcuts import render

def token_balances(request): api_key = 'your_covalent_api_key' wallet_address = 'your_wallet_address' chain_id = 'alphabill_chain_id'

```
url = f'https://api.covalenthq.com/v1/{chain_id}/address/{wallet_address}/balances_v2/?key={a
response = requests.get(url)
data = response.json()

context = {
    'token_balances': data['data']['items']
}

return render(request, 'token_balances.html', context)
```

`

Step 4: Real-Time Notifications

- Use Covalent's WebSocket API for real-time notifications.

Example: Setting Up WebSocket for Real-Time Data

- Install WebSocket client.

```
pip install websocket-client
```

- Create a WebSocket client to listen for blockchain events.

`import websocket import json

def on_message(ws, message): data = json.loads(message) print(f"New Event: ")

def on_error(ws, error): print(f"Error: ")

def on_close(ws): print("Connection closed")

def on_open(ws): print("Connection opened")

ws = websocket.WebSocketApp('wss://api.covalenthq.com/v1/YOUR_WEBSOCKET_ENDPOINT', on_message=on_message, on_error=on_error, on_close=on_close) ws.on_open = on_open ws.run_forever() `

Step 5: Display Real-Time Notifications

- Integrate WebSocket notifications into your Django LMS.

Example: Django View for Real-Time Notifications

- Create a Django view to handle WebSocket notifications.

`from django.http import JsonResponse

def real_time_notifications(request): # Logic to handle WebSocket notifications return JsonResponse({'status': 'Listening for events'}) `

Resources:

- Covalent API Documentation
- Django Documentation
- WebSocket Client for Python

F. Decentralized Storage

Must-Have Features for the MVP:

1. *File Upload:* Enable users to upload files to the decentralized storage.
2. *File Retrieval:* Allow users to retrieve and view stored files.
3. *Security:* Ensure that files are securely stored and only accessible to authorized users.

Step-by-Step Integration Procedure:

Step 1: Set Up IPFS

- Install IPFS on your server or use an IPFS service provider like Infura.
- Install IPFS Locally:

```
wget https://dist.ipfs.tech/go-ipfs/v0.8.0/go-ipfs_v0.8.0_linux-amd64.tar.gz tar -xvzf go-ipfs_v0.8.0_linux-amd64.tar.gz cd go-ipfs sudo bash install.sh ipfs init ipfs daemon
```

Step 2: Integrate IPFS with Django LMS

Example: File Upload to IPFS

- Create a Django form for file upload and a view to handle the upload. File Upload Form:

`from django import forms

class FileUploadForm(forms.Form): file = forms.FileField() ` File Upload View:

`import ipfshttpclient from django.shortcuts import render from django.http import HttpResponse from .forms import FileUploadForm

def upload_file(request): if request.method == 'POST': form = FileUploadForm(request.POST, request.FILES) if form.is_valid(): file = request.FILES['file'] with ipfshttpclient.connect() as client: res = client.add(file) return HttpResponse(f"File uploaded successfully: {res['Hash']}") else: form = FileUploadForm() return render(request, 'upload_file.html', {'form': form}) ` Upload Template:

`

{% csrf_token %} {{ form.as_p }} Upload

` ### Step 3: Retrieve Files from IPFS

File Retrieval View:

```
def retrieve_file(request, file_hash): with ipfshttpclient.connect() as client: file_data = client.cat(file_hash) response = HttpResponse(file_data, content_type='application/octet-stream') response['Content-Disposition'] = f'attachment; filename="{file_hash}"' return response
```

Step 4: Secure Access to Files

- Implement authentication and authorization to ensure that only authorized users can upload and retrieve files.

Example: Secure File Upload and Retrieval

`from django.contrib.auth.decorators import login_required

@login_required def upload_file(request): if request.method == 'POST': form = FileUploadForm(request.POST, request.FILES) if form.is_valid(): file = request.FILES['file'] with ipfshttpclient.connect() as client: res = client.add(file) return HttpResponse(f"File uploaded successfully: {res['Hash']}") else: form = FileUploadForm() return render(request,

'upload_file.html', {'form': form})

@login_required def retrieve_file(request, file_hash): with ipfshttpclient.connect() as client: file_data = client.cat(file_hash) response = HttpResponse(file_data, content_type='application/octet-stream') response['Content-Disposition'] = f'attachment; filename="" return response `

Resources:

- IPFS Documentation
- IPFS HTTP Client for Python
- Django Authentication

G. Decentralized Identity (DID)

Must-Have Features for the MVP:

1. Identity Creation: Allow users to create and manage their decentralized identities.
2. Verifiable Credentials: Enable issuance and verification of verifiable credentials.
3. Secure Data Storage: Store identity data securely on the decentralized network.

Step-by-Step Integration Procedure:

Step 1: Set Up Ceramic Network

- Set up a Ceramic node or use a service provider like Ceramic Network.

Install Ceramic CLI:

```
npm install -g @ceramicnetwork/cli ceramic daemon
```

Step 2: Integrate Ceramic with Django LMS

Example: Identity Creation

- Create a Django form for identity creation and a view to handle the creation.

Identity Creation Form:

`from django import forms

class IdentityCreationForm(forms.Form): username = forms.CharField(max_length=100) ` Identity Creation View:

`import ceramic from django.shortcuts import render from django.http import HttpResponse from .forms import IdentityCreationForm

def create_identity(request): if request.method == 'POST': form = IdentityCreationForm(request.POST) if form.is_valid(): username = form.cleaned_data['username'] # Create identity on Ceramic ceramic_client = ceramic.Ceramic() did = ceramic_client.create_did(username) return HttpResponse(f"Identity created successfully: ") else: form = IdentityCreationForm() return render(request, 'create_identity.html', {'form': form}) ` Identity Creation Template:

`

{% csrf_token %} {{ form.as_p }} Create Identity

` Identity Creation Template:

`

{% csrf_token %} {{ form.as_p }} Create Identity

` ### Step 3: Issue and Verify Verifiable Credentials

Example: Issuing Verifiable Credentials

- Create a Django view to issue verifiable credentials.

Issue Credentials View:

```
def issue_credentials(request): if request.method == 'POST': username =
request.POST.get('username') credential_data = request.POST.get('credential_data') #
Issue credentials on Ceramic ceramic_client = ceramic.Ceramic() credentials =
ceramic_client.issue_credentials(username, credential_data) return
HttpResponse(f"Credentials issued successfully: {credentials}") return
render(request, 'issue_credentials.html')
```

Issue Credentials Template:

`

{% csrf_token %} Username: Credential Data: Issue Credentials

` Example: Verifying Verifiable Credentials

- Create a Django view to verify credentials.

Verify Credentials View:

```
def verify_credentials(request): if request.method == 'POST': credentials =
request.POST.get('credentials') # Verify credentials on Ceramic ceramic_client =
ceramic.Ceramic() is_valid = ceramic_client.verify_credentials(credentials) return
HttpResponse(f"Credentials valid: {is_valid}") return render(request,
'verify_credentials.html')
```

Verify Credentials Template:

`

{% csrf_token %} Credentials: Verify Credentials

` ### Step 4: Secure Data Storage

- Store identity data securely using Ceramic and IPFS.

Resources:

- Ceramic Network Documentation > OrbisDB
-

- Django Documentation
- IPFS Documentation