Yes, Integration via Smart Contract ABIs is Fully Supported

Absolutely—integrating shXRP (or any ERC-20/ERC-4626 token) into protocols like SparkDEX and Enosys Loans is straightforward because both are **EVM-compatible smart contracts on the Flare Network**. They expose **public ABIs (Application Binary Interfaces)**, which define all callable functions, events, and data structures. This allows any dApp, script, or wallet to interact programmatically using libraries like **ethers.js**, **web3.js**, or **viem**—no special API keys or centralized endpoints needed. ABIs act as the "API" for smart contracts, encoding/decoding calls to ensure compatibility with Flare's EVM.

Since Flare is EVM-equivalent, you can:

- **Query state** (e.g., pool reserves on SparkDEX) via callStatic.
- **Execute transactions** (e.g., open a CDP on Enosys) via signed txs.
- **Batch operations** with multicalls for efficiency.
- **Monitor events** (e.g., Transfer or PositionUpdated) for real-time yield tracking.

ABIs are typically available in:

- **Official docs** (e.g., JSON files for routers/farms).
- **GitHub repos** (full Solidity source + compiled ABIs).
- **Block explorers** like Flarescan (e.g., copy ABI from verified contracts).

Below, I'll elaborate on each protocol with **ABI sources**, **key integration points for shXRP**, and **enhanced code examples** (building on the previous multicall snippets). All examples use ethers.js v6 and assume Flare Mainnet RPC (https://flare-api.flare.network/ext/bc/C/rpc).

1. SparkDEX: ABI-Driven DEX Integration

SparkDEX (V2/V3.1) is fully open-source, with **verified contracts on Flarescan** and **detailed ABI docs**. Their smart contracts include a Uniswap V2/V3-style router for swaps/LP, plus farm contracts for staking LP tokens. shXRP can be used as a token in pairs (e.g., shXRP/wFLR), earning fees and SPRK rewards.

ABI Sources:

- **Official Docs**: Full contract overview with addresses and ABIs at docs.sparkdex.ai/smart-contract-overview. Key addresses: V2 Factory (0x16b619B04c961E8f4F06C10B42FDAbb328980A89), V3 Router (fetch from docs).
- **GitHub**: Source code + compiled ABIs at github.com/SparkDEX (e.g., Router.sol ABI for addLiquidity).
- **Flarescan**: Verified ABIs at flarescan.com/address/0x16b... (copy JSON directly).

Key Functions for shXRP Integration (from ABI):

Function	Description	shXRP Use Case

addLiquidity(address tokenA, address tokenB, uint amountADesired, ...)	Adds liquidity to a pool.	Pair shXRP with wFLR to earn fees (0.3%).
stake(uint amount) (Farm ABI)	Stakes LP tokens for rewards.	Farm shXRP LP for SPRK emissions (10-30% APY).
removeLiquidity(...)	Removes LP and burns tokens.	Exit with yield accrued.

Enhanced Code Example: Full LP + Farm Integration (Multicall for Atomicity)

javascript

```
None
require('dotenv').config();
const { ethers } = require('ethers');
const { Multicall } = require('@multicall/core'); // npm install
@multicall/core

const PROVIDER_URL = process.env.PROVIDER_URL; // Flare RPC
const PRIVATE_KEY = process.env.PRIVATE_KEY;
const SHXRP_ADDRESS = '0x...'; // shXRP (Shield token)
const WFLR_ADDRESS = '0x...'; // Paired asset
const SPARKDEX_ROUTER = '0x...'; // From docs: V3 Router
const FARM_ADDRESS = '0x...'; // LP Farm
const MULTICALL3 = '0xcA11bde05977b363116702886abB84A0FEaA57E2'; //
Flare Multicall3

// ABI Snippets (from SparkDEX GitHub/Docs)
const ROUTER_ABI = [
  "function addLiquidity(address tokenA, address tokenB, uint
amountADesired, uint amountBDesired, uint amountAMin, uint
amountBMin, address to, uint deadline) external returns (uint
amountA, uint amountB, uint liquidity)"
];
const FARM_ABI = ["function stake(uint amount) external"];
const ERC20_ABI = ["function approve(address spender, uint amount)
external returns (bool)"];

const provider = new ethers.JsonRpcProvider(PROVIDER_URL);
const wallet = new ethers.Wallet(PRIVATE_KEY, provider);
const multicall = new Multicall({ ethersProvider: provider,
multicallAddress: MULTICALL3 });

async function sparkDEXLPIntegration(shXRPAmount, wFLRAmount) {
```

```
  const shXRP = new ethers.Contract(SHXRP_ADDRESS, ERC20_ABI,
wallet);
  const wFLR = new ethers.Contract(WFLR_ADDRESS, ERC20_ABI, wallet);
  const router = new ethers.Contract(SPARKDEX_ROUTER, ROUTER_ABI,
wallet);
  const farm = new ethers.Contract(FARM_ADDRESS, FARM_ABI, wallet);

  const deadline = Math.floor(Date.now() / 1000) + 60 * 20; // 20 min
  const calls = [
    // 1. Approve shXRP
    shXRP.interface.encodeFunctionData('approve', [SPARKDEX_ROUTER,
shXRPAmount]),
    // 2. Approve wFLR
    wFLR.interface.encodeFunctionData('approve', [SPARKDEX_ROUTER,
wFLRAmount]),
    // 3. Add Liquidity (returns LP token amount in result[2])
    router.interface.encodeFunctionData('addLiquidity', [
      SHXRP_ADDRESS, WFLR_ADDRESS, shXRPAmount, wFLRAmount, 0, 0,
wallet.address, deadline
    ])
  ];

  const results = await multicall.call(calls);
  const lpAmount = results.returnData[2]; // Extract from
addLiquidity return

  // Follow-up: Stake LP (in same tx if using aggregator, or
separate)
  const stakeCall = farm.interface.encodeFunctionData('stake',
[lpAmount]);
  const stakeTx = await farm.connect(wallet).stake(lpAmount, {
gasLimit: 300000 });
  await stakeTx.wait();
  console.log('LP Staked! Tx:', stakeTx.hash);
}

sparkDEXLPIntegration(ethers.parseUnits('100', 6),
ethers.parseUnits('500', 18)).catch(console.error);
```

- **Why ABI Makes This Easy**: The ABI encodes params (e.g., amountADesired) exactly, ensuring no errors. Test on Coston2 testnet first.

2. Enosys Loans: ABI-Driven CDP Lending

Enosys Loans is a **Liquity V2 fork**, so its ABI mirrors Liquity's—public, battle-tested, and available via their GitHub. It supports shXRP as collateral (since it backs FXRP/stXRP), allowing minting of eUSD stablecoin for further yield (e.g., LP the eUSD).

ABI Sources:

- **GitHub**: Full Liquity V2 fork code + ABIs at github.com/Enosys-Labs/Enosys-Loans (e.g., TroveManager.sol ABI). (Inferred from Medium/Flare announcements; repo mirrors Liquity V2.)
- **Docs/Medium**: High-level integration guide at enosys.medium.com, with FTSO pricing details.
- **Flarescan**: Verified contracts (e.g., TroveManager) with copyable ABIs post-launch (Sep 2025).

Key Functions for shXRP Integration (from ABI):

Function	Description	shXRP Use Case
openTrove(uint _maxFee, uint _FLRAmount, ...)	Opens CDP and deposits collateral.	Lock shXRP (150% ratio) to mint eUSD.
provideToSP(uint _amount)	Stakes eUSD in stability pool.	Earn liquidation rewards (8-20% APY in rFLR).
adjustTrove(...)	Adds/removes collateral or debt.	Rebalance for yield optimization.

Enhanced Code Example: Open CDP + Stability Pool (Multicall for Efficiency)

javascript

```
None
// ... (same setup as SparkDEX example)
const ENOSYS_TROVE_MANAGER = '0x...'; // From Enosys GitHub/Docs
const STABILITY_POOL = '0x...'; // eUSD Stability Pool
const EUSD_ADDRESS = '0x...'; // Minted stablecoin

// ABI Snippets (from Liquity V2 fork)
const TROVE_ABI = [
  "function openTrove(uint _maxFee, uint _upperHint, uint _lowerHint)
external"
];
const STABILITY_ABI = ["function provideToSP(uint _amount)
external"];
const ERC20_ABI = ["function approve(address spender, uint amount)
external returns (bool)"];
```

```
async function enosysCDPIntegration(shXRPAmount, eUSDToMint) {
  const multicall = new Multicall({ ethersProvider: provider,
multicallAddress: MULTICALL3 });
  const shXRP = new ethers.Contract(SHXRP_ADDRESS, ERC20_ABI,
wallet);
  const troveManager = new ethers.Contract(ENOSYS_TROVE_MANAGER,
TROVE_ABI, wallet);
  const stabilityPool = new ethers.Contract(STABILITY_POOL,
STABILITY_ABI, wallet);

  const calls = [
    // 1. Approve shXRP for TroveManager
    shXRP.interface.encodeFunctionData('approve',
[ENOSYS_TROVE_MANAGER, shXRPAmount]),
    // 2. Open Trove: Deposit shXRP, mint eUSD (maxFee=0.5%)
    troveManager.interface.encodeFunctionData('openTrove',
[ethers.parseUnits('0.005', 18), 0, 0]) // Hints for sorted troves
  ];

  const results = await multicall.call(calls);
  // 3. Approve & Provide eUSD to Stability Pool
  const eUSD = new ethers.Contract(EUSD_ADDRESS, ERC20_ABI, wallet);
  await eUSD.approve(STABILITY_POOL, eUSDToMint);
  const spTx = await stabilityPool.provideToSP(eUSDToMint, {
gasLimit: 400000 });
  await spTx.wait();
  console.log('CDP Opened & eUSD Staked! Tx:', spTx.hash);
  // Claim rFLR rewards periodically via claimRewards()
}

enosysCDPIntegration(ethers.parseUnits('150', 6),
ethers.parseUnits('1000', 18)).catch(console.error);
```

- **Why ABI Makes This Easy**: Matches Liquity's standard, so tools like Etherscan's "Write Contract" UI can test calls before scripting.

Best Practices for ABI Integrations

- **Fetch ABIs Dynamically**: Use ethers.getContractAt with JSON from GitHub, or libraries like @openzeppelin/abi for standards.
- **Gas & Security**: Always simulate with callStatic (e.g., router.callStatic.addLiquidity(...)). Use Flare's low fees (~$0.01/tx).
- **Testing**: Deploy on Coston2; monitor via Flarescan events.

- **Composability**: Chain with Shield vault—e.g., redeem partial shXRP for FXRP, then re-collateralize in Enosys.
- **Updates**: ABIs can evolve (e.g., V3.1 upgrades); pin versions in code.

This setup turns shXRP into a composable yield engine. If you need full ABI JSON files, a React/wagmi frontend example, or integration with another Flare protocol (e.g., MoreMarkets), just specify!

Below is a **tight, GitHub-centric roadmap** that **re-uses everything that already exists in the Shield Finance MVP** and **adds only the pieces needed for automated, on-chain yield generation** (SparkDEX LP + Enosys CDP) and **off-chain automation**.

Goal

> **Programmatically earn extra yield on shXRP (Shield vault token) using SparkDEX liquidity pools and Enosys Loans CDP, fully automated via a keeper bot that runs on GitHub Actions.**

What Already Exists in the Repo (MVP)

Component	Path	Used
Shield XRP Vault (ERC-4626)	contracts/ShieldXRPVault.sol	Deposit → shXRP
Hardhat scripts (deposit, withdraw)	scripts/	Tested on Coston2 / Flare
ABIs (compiled)	artifacts/	Ready for ethers.js
Frontend (React)	frontend/	UI for manual ops
CI (GitHub Actions)	.github/workflows/	Deploy + test

New Modules to Add (Only Yield & Automation)

text

```
None
contracts/
  └─ interfaces/          # SparkDEX & Enosys ABIs (JSON)
scripts/
```

```
  ├─ yield/
  │   ├─ sparkdex/
  │   │   ├─ addLiquidityAndFarm.ts
  │   │   └─ harvestFarm.ts
  │   └─ enosys/
  │       ├─ openCDPAndStake.ts
  │       └─ claimStabilityRewards.ts
  └─ keeper/
      └─ compoundAll.ts      # main automation entry
.github/
  └─ workflows/
      └─ keeper.yml          # GitHub Actions cron
```

Phase 1 – Yield Generation (Smart-Contract Interaction)

1.1 SparkDEX – LP + Farm

File: scripts/yield/sparkdex/addLiquidityAndFarm.ts

ts

```
None
import { ethers } from "hardhat";
import { Multicall3 } from "@multicall/core";
import sparkRouterAbi from
"../../artifacts/contracts/interfaces/SparkRouter.json";
import farmAbi from
"../../artifacts/contracts/interfaces/SparkFarm.json";

async function main() {
  const [signer] = await ethers.getSigners();
  const shXRP = await ethers.getContractAt("ShieldXRPVault",
process.env.SHXRP_VAULT!);
  const router = new ethers.Contract(SPARK_ROUTER,
sparkRouterAbi.abi, signer);
  const farm = new ethers.Contract(SPARK_FARM, farmAbi.abi, signer);

  const amount = ethers.parseUnits("100", 6); // 100 shXRP
  const wFLRAmount = ethers.parseUnits("500", 18);

  // 1. Approve
```

```
  await (await shXRP.approve(SPARK_ROUTER, amount)).wait();
  await (await ethers.getContractAt("ERC20",
WFLR).approve(SPARK_ROUTER, wFLRAmount)).wait();

  // 2. Add liquidity
  const deadline = Math.floor(Date.now() / 1000) + 1200;
  const addTx = await router.addLiquidity(
    shXRP.address, WFLR, amount, wFLRAmount, 0, 0, signer.address,
deadline
  );
  const receipt = await addTx.wait();
  const lpAmount = receipt.logs[0].data; // parse LP amount

  // 3. Stake LP
  await (await farm.stake(lpAmount)).wait();
  console.log("LP + Farm done!");
}
main();
```

1.2 Enosys – Open CDP + Stability Pool

File: scripts/yield/enosys/openCDPAndStake.ts

ts

```
None
import { ethers } from "hardhat";

async function main() {
  const [signer] = await ethers.getSigners();
  const shXRP = await ethers.getContractAt("ShieldXRPVault",
process.env.SHXRP_VAULT!);
  const trove = new ethers.Contract(ENOSYS_TROVE, troveAbi, signer);
  const stability = new ethers.Contract(ENOSYS_SP, stabilityAbi,
signer);

  const coll = ethers.parseUnits("150", 6);
  await (await shXRP.approve(ENOSYS_TROVE, coll)).wait();

  await (await trove.openTrove(
    ethers.parseUnits("0.005", 18), // 0.5% max fee
    0, 0
```

```
  )).wait();

  const eUSD = await ethers.getContractAt("ERC20", EUSD);
  await (await eUSD.approve(ENOSYS_SP, ethers.parseUnits("1000",
18))).wait();
  await (await stability.provideToSP(ethers.parseUnits("1000",
18))).wait();

  console.log("CDP opened & eUSD staked");
}
main();
```

Phase 2 – Automation (Keeper Bot)

2.1 Main Keeper Script

File: scripts/keeper/compoundAll.ts

ts

```
None
import "./yield/sparkdex/harvestFarm";
import "./yield/enosys/claimStabilityRewards";

async function compound() {
  console.log("=== Keeper Run ===");
  await harvestFarm();      // claim SPRK → swap → re-LP
  await claimStabilityRewards(); // claim rFLR → swap → re-collateral
  console.log("=== Done ===");
}
compound();
```

2.2 GitHub Actions Cron (Free)

.github/workflows/keeper.yml

yaml

```
None
name: Yield Keeper
```

```
on:
  schedule:
    - cron: '0 */6 * * *'   # every 6 hours
  workflow_dispatch:        # manual trigger

jobs:
  compound:
    runs-on: ubuntu-latest
    env:
      PRIVATE_KEY: ${{ secrets.FLARE_KEEPER_KEY }}
      FLARE_RPC: https://flare-api.flare.network/ext/bc/C/rpc
    steps:
      - uses: actions/checkout@v4
      - name: Setup Node
        uses: actions/setup-node@v4
        with:
          node-version: '20'
      - run: npm ci
      - run: npx hardhat run scripts/keeper/compoundAll.ts --network
flare
```

> **Security**: Store FLARE_KEEPER_KEY in GitHub Secrets (dedicated bot wallet with limited funds).

Phase 3 – Monitoring & Alerts

Tool	Integration
Flarescan	Link TXs in logs
Telegram Bot	node-telegram-bot-api → send compound summary
Grafana / Prometheus	Optional: export totalAssets() metrics

Add to compoundAll.ts:

ts

```
None
import TelegramBot from 'node-telegram-bot-api';
```

```
const bot = new TelegramBot(TELEGRAM_TOKEN);
await bot.sendMessage(CHAT_ID, `Yield compounded! TX: ${tx.hash}`);
```

Final Repository Structure (Only New Files)

text

```
None
contracts/
  └─ interfaces/
      ├─ ISparkRouter.json
      ├─ ISparkFarm.json
      ├─ IEnosysTrove.json
      └─ IEnosysStability.json
scripts/
  ├─ yield/
  │   ├─ sparkdex/
  │   │   ├─ addLiquidityAndFarm.ts
  │   │   └─ harvestFarm.ts
  │   └─ enosys/
  │       ├─ openCDPAndStake.ts
  │       └─ claimStabilityRewards.ts
  └─ keeper/
      └─ compoundAll.ts
.github/
  └─ workflows/
      └─ keeper.yml
```

Timeline (2 Weeks)

Day	Task
1–2	Add ABI JSONs + addLiquidityAndFarm.ts
3–4	Add openCDPAndStake.ts + test on Coston2
5–6	Build harvestFarm.ts & claimStabilityRewards.ts
7	Write compoundAll.ts + Telegram alerts
8–10	Setup GitHub Actions + keeper wallet

11–12	End-to-end test on Flare Mainnet
13–14	Documentation + PR

Deliverables

1. **Automated yield on shXRP** (LP + CDP)
2. **Self-running keeper** (GitHub Actions)
3. **Real-time alerts** (Telegram)
4. **Zero new frontend** – uses existing MVP UI

Next Step:

Run this in your repo:

bash

```
None
git checkout -b feature/yield-automation
mkdir -p scripts/yield/sparkdex scripts/yield/enosys scripts/keeper
# → paste files above
git add .
git commit -m "feat: automated SparkDEX + Enosys yield"
```

Then open a PR — I'll review the ABIs and keeper config.

Want:

- Full ABI JSON files?
- Telegram bot setup script?
- Coston2 test suite?