Solana DEX/AMM MVP - Solution Architecture

Project Overview

Client Need: A simple DEX with AMM functionality for creating liquidity pools (LPs) with Solana network tokens, with Jupiter Aggregator route compatibility.

Focus: Minimal viable product with lowest possible fees

Requirements Summary

Core Features

1. **Liquidity Pool Creation** - Create concentrated LPs between SPL tokens
2. **Simple AMM** - Basic automated market maker with:
   - Add liquidity
   - Remove liquidity
   - Token swaps
3. **Jupiter Routes Preparation** - Structure compatible with Jupiter Aggregator
4. **Web Interface** - Lightweight frontend with wallet connection

Wallet Support

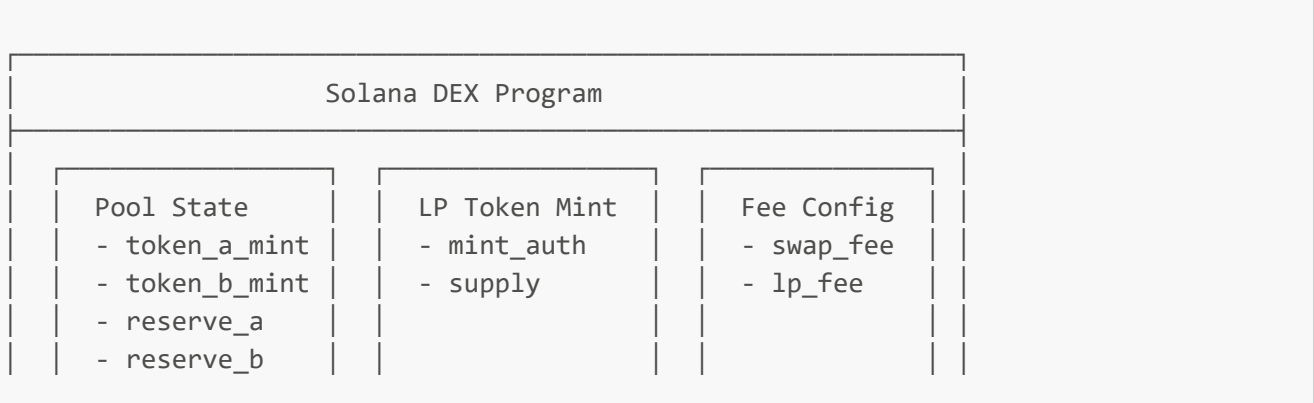
- Phantom Wallet
- Solflare Wallet

Technical Stack

- **Blockchain:** Solana (Devnet for testing, Mainnet for production)
- **Smart Contracts:** Anchor Framework (Rust)
- **Frontend:** React.js with Solana Wallet Adapter
- **Hosting:** Static hosting (Vercel/Netlify/Cloudflare Pages - free)

Architecture Design

1. On-Chain Program (Smart Contract)



```
│  └──────────────────┘  └──────────────────┘  └──────────────┘ │
├──────────────────────────────────────────────────────────────┤
│  Instructions:                                               │
│  ├── initialize_pool(token_a, token_b, fee_rate)             │
│  ├── add_liquidity(amount_a, amount_b, min_lp_tokens)        │
│  ├── remove_liquidity(lp_tokens, min_a, min_b)               │
│  └── swap(amount_in, min_amount_out, direction)              │
└──────────────────────────────────────────────────────────────┘
```

2. AMM Model: Constant Product (x * y = k)

The simplest and most proven AMM model:

- **Formula:** `reserve_a * reserve_b = constant`
- **Swap Price:** `price = reserve_b / reserve_a`
- **Slippage:** Built-in via the curve

Why this model:

- Simple to implement and audit
- Battle-tested (used by Uniswap V2, Raydium)
- Low computational cost = low fees
- Jupiter compatible

3. Account Structure (Cost Optimized)

```
Pool Account (PDA):
├── Discriminator: 8 bytes
├── Token A Mint: 32 bytes
├── Token B Mint: 32 bytes
├── Token A Vault: 32 bytes
├── Token B Vault: 32 bytes
├── LP Token Mint: 32 bytes
├── Fee Rate: 2 bytes (u16, basis points)
├── Bump: 1 byte
└── Total: ~171 bytes + padding = ~200 bytes

Rent Cost: ~0.002 SOL per pool
```

4. Frontend Architecture

```
┌──────────────────────────────────────────────────────────────┐
│                    React Frontend                            │
├──────────────────────────────────────────────────────────────┤
│  ┌──────────────────────────────────────────────────────┐    │
│  │              Solana Wallet Adapter                   ||
│  │  ├── Phantom Provider                                ||
│  │  └── Solflare Provider                               ||
```

```
│  └──────────────────────────────────────────────────────┘  │
├────────────────────────────────────────────────────────────┤
│  Pages:                                                    │
│  ├── /              → Pool List & Stats                    │
│  ├── /swap          → Token Swap Interface                 │
│  ├── /pool/:id      → Pool Details                         │
│  └── /liquidity     → Add/Remove Liquidity                 │
├────────────────────────────────────────────────────────────┤
│  Components:                                               │
│  ├── WalletButton   → Connect/Disconnect                   │
│  ├── TokenSelector  → Select tokens with balances          │
│  ├── SwapCard       → Swap interface                       │
│  ├── LiquidityCard  → Add/Remove liquidity                 │
│  └── PoolList       → Display available pools              │
└────────────────────────────────────────────────────────────┘
```

Jupiter Integration Strategy

For the MVP with limited budget, we implement **Jupiter-Ready** architecture:

Phase 1 (MVP - Included)

- Standard instruction format compatible with Jupiter
- Proper account structure for external calls
- Pool discovery accounts

Phase 2 (Future - Not in MVP)

- Full Jupiter SDK integration
- Routing API registration
- Real-time price feeds

Jupiter Compatibility Requirements:

1. Use standard SPL Token accounts
2. Implement `swap` instruction with predictable signature
3. Provide on-chain pool state for price calculation
4. Use PDAs for pool addresses (deterministic)

Fee Structure

On-Chain Fees (Solana Network)

Operation	Estimated Cost
Create Pool	0.01-0.02 SOL
Add Liquidity	0.0001-0.0005 SOL

Operation	Estimated Cost
Remove Liquidity	0.0001-0.0005 SOL
Swap	0.00001-0.0001 SOL

DEX Fees (Configurable)

Fee Type	Default	Range
Swap Fee	0.3%	0.1% - 1%
LP Fee Share	100%	Configurable

Security Considerations

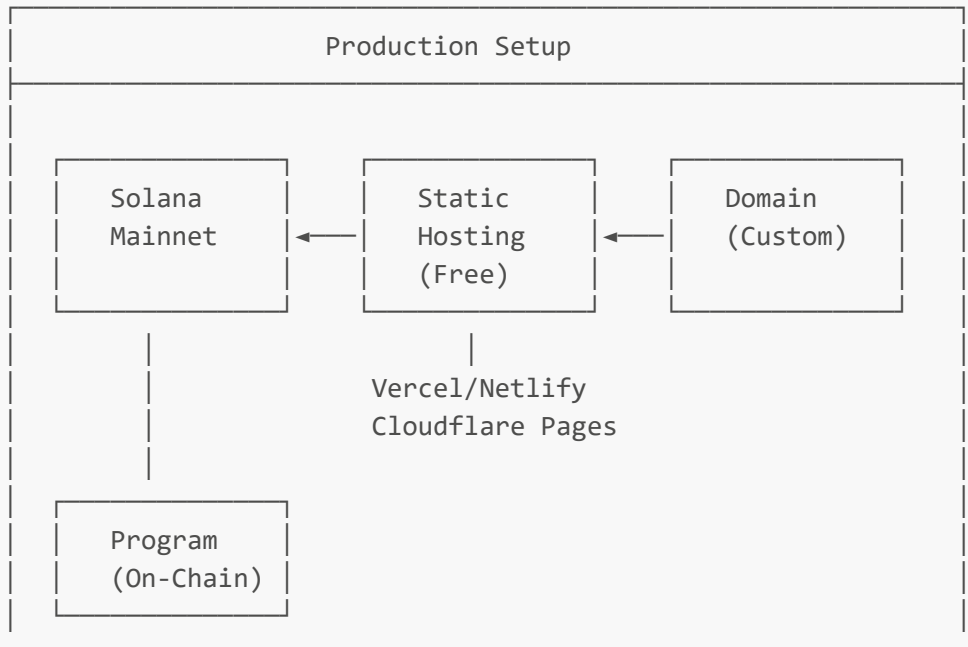
MVP Security Measures

1. **Overflow Protection** - Using checked math operations
2. **Slippage Protection** - Minimum output amount checks
3. **Reentrancy Prevention** - Anchor's built-in protections
4. **Authority Checks** - Proper signer validations
5. **PDA Validation** - Verify account derivations

Limitations (MVP Scope)

- No formal audit (budget constraint)
- Basic testing only
- Recommended for small amounts initially

Deployment Architecture



```
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

File Structure

```
solana-dex-mvp/
├── .env                          # Centralized configuration
├── .env.example                  # Configuration template
├── anchor/                       # On-chain program
│   ├── Anchor.toml
│   ├── Cargo.toml
│   └── programs/
│       └── dex/
│           ├── Cargo.toml
│           └── src/
│               ├── lib.rs        # Program entry point
│               ├── state.rs      # Account structures
│               ├── instructions/ # Instruction handlers
│               │   ├── mod.rs
│               │   ├── initialize_pool.rs
│               │   ├── add_liquidity.rs
│               │   ├── remove_liquidity.rs
│               │   └── swap.rs
│               ├── errors.rs     # Custom errors
│               └── constants.rs  # Program constants
├── frontend/                     # React application
│   ├── package.json
│   ├── src/
│   │   ├── App.tsx
│   │   ├── config/
│   │   │   └── env.ts           # Frontend config from .env
│   │   ├── components/
│   │   ├── hooks/
│   │   ├── utils/
│   │   └── idl/                 # Generated IDL
│   └── public/
├── tests/                        # Integration tests
│   └── dex.ts
├── scripts/                      # Deployment scripts
│   ├── deploy.sh
│   └── create-pool.ts
└── docs/                         # Documentation
    ├── SOLUTION_ARCHITECTURE.md
    ├── IMPLEMENTATION_GUIDE.md
    └── DEPLOYMENT.md
```

Technology Decisions

Component	Choice	Reason
Smart Contract Framework	Anchor	Industry standard, safe, well-documented
AMM Model	Constant Product	Simple, proven, Jupiter compatible
Frontend Framework	React + Vite	Fast, lightweight, good DX
Wallet Integration	@solana/wallet-adapter	Official, supports Phantom + Solflare
Styling	Tailwind CSS	Rapid development, small bundle
State Management	React Query	Efficient RPC data fetching

Success Criteria

MVP Deliverables

- [ ] On-chain DEX program deployed to Devnet
- [ ] Create liquidity pool functionality
- [ ] Add/Remove liquidity operations
- [ ] Token swap functionality
- [ ] Web interface with wallet connection
- [ ] Phantom and Solflare support
- [ ] Basic documentation
- [ ] Mainnet deployment guide

Quality Metrics

- All core functions working on Devnet
- Transaction fees under 0.001 SOL for swaps
- Frontend loads in under 3 seconds
- Mobile-responsive design