Jupiter Swap DApp

Technical Documentation & README

Advanced SOL/USDC Trading Platform

Production-Ready DApp with Advanced Optimizations

Framework: Next.js 14 (App Router)	**Styling:** Tailwind CSS + shadcn/ui
Language: TypeScript 5.3+	**State:** Zustand + React Query
Blockchain: Solana Mainnet	**Testing:** Jest + Testing Library
API: Jupiter v6	**Deployment:** Vercel + GitHub Actions

Key Features

Dynamic Slippage Optimization
Smart Priority Fee Calculation
MEV Protection Integration
Multi-RPC Failover System
Real-time Balance Tracking
Comprehensive Error Handling
Performance Monitoring (Sentry)
Mobile-Responsive Design

Developed by: Kamel (@treizeb__)
Company: DeAura.io
Period: July 2025

Contents

1 Quick Start

1.1 Prerequisites

Required Software:

- Node.js 18.17+ or 20.0+
- npm 9.0+ or yarn 1.22+ or pnpm 8.0+
- Git 2.30+
- A Solana wallet (Phantom, Solflare, etc.)

1.2 Installation

```
# Clone the repository
git clone https://github.com/deaura-io/jupiter-swap-nextjs.git
cd jupiter-swap-nextjs

# Install dependencies
npm install
# or
yarn install
# or
pnpm install

# Copy environment variables
cp .env.example .env.local

# Start development server
npm run dev
# or
yarn dev
# or
pnpm dev
```

Listing 1: Installation Commands

1.3 Environment Configuration

```
# Application Configuration
NEXT_PUBLIC_APP_NAME="Jupiter Swap DApp"
NEXT_PUBLIC_APP_DESCRIPTION="Advanced SOL/USDC Trading Platform"
NEXT_PUBLIC_ENVIRONMENT="development"

# Solana Configuration
NEXT_PUBLIC_SOLANA_NETWORK="mainnet-beta"
NEXT_PUBLIC_RPC_URL="https://api.mainnet-beta.solana.com"

# API Keys (Production)
NEXT_PUBLIC_HELIUS_API_KEY="d94d81dd-f2a1-40f7-920d-0dfaf3aaf032"
NEXT_PUBLIC_ALCHEMY_API_KEY="UvOk23LRlqGz1m58VCEd3PJ2ZOX2h9KM"

# RPC Endpoints with Failover
NEXT_PUBLIC_FALLBACK_RPC_URLS='[
  "https://eclipse.helius-rpc.com/",
  "https://api.mainnet-beta.solana.com",
  "https://solana-api.projectserum.com"
]'

# Jupiter API Configuration
NEXT_PUBLIC_JUPITER_API_BASE="https://quote-api.jup.ag"
NEXT_PUBLIC_JUPITER_API_VERSION="v6"

# Optimization Settings
NEXT_PUBLIC_ENABLE_OPTIMIZATIONS="true"
NEXT_PUBLIC_DEFAULT_SLIPPAGE_BPS="50"
NEXT_PUBLIC_MAX_SLIPPAGE_BPS="300"
NEXT_PUBLIC_ENABLE_MEV_PROTECTION="true"

# Fee Recovery Configuration
NEXT_PUBLIC_ENABLE_FEE_RECOVERY="true"
NEXT_PUBLIC_FEE_RECOVERY_PERCENTAGE="25"
NEXT_PUBLIC_SERVICE_WALLET="DeAura1234567890123456789012345678901234"

# Monitoring & Analytics
NEXT_PUBLIC_SENTRY_DSN="your_sentry_dsn_here"
NEXT_PUBLIC_ENABLE_ANALYTICS="true"

# Development Settings
NEXT_PUBLIC_ENABLE_DEVTOOLS="true"
NEXT_PUBLIC_LOG_LEVEL="info"
```

Listing 2: Environment Variables (.env.local)

2 Architecture Overview

2.1 Project Structure

```
jupiter-swap-nextjs/
          src/
                app/                    # Next.js App Router
                      layout.tsx         # Root layout
                      page.tsx           # Home page
                      privacy/           # Privacy policy
                      terms/             # Terms of service
                      api/               # API routes
                components/            # React components
                      swap/              # Swap-related components
                      wallet/            # Wallet components
                      ui/                # shadcn/ui components
                      layout/            # Layout components
                      analytics/         # Analytics components
                      providers/         # Context providers
                services/              # Business logic
                      jupiter.ts         # Jupiter API integration
                      solana.ts          # Solana blockchain
                      swap.ts            # Swap orchestration
                      optimization.ts    # Trading optimizations
                      rpc-manager.ts     # RPC management
                      errors.ts          # Error handling
                      feeRecovery.ts     # Fee recovery logic
                hooks/                 # Custom React hooks
                store/                 # Zustand stores
                types/                 # TypeScript definitions
                utils/                 # Utility functions
                constants/             # Application constants
                styles/                # Global styles
          public/                    # Static assets
```

```
        docs/                       # Documentation
        tests/                      # Test files
        config files                # Configuration files
```

Listing 3: Project Directory Structure

2.2 Technology Stack

Category	Technology	Version	Purpose
3*Frontend	Next.js	14.2.0	React framework
	TypeScript	5.3+	Type safety
	Tailwind CSS	3.4.0	Styling
4*Blockchain	@solana/web3.js	1.91.4	Solana integration
	@solana/spl-token	0.4.1	Token operations
	@jup-ag/react-hook	6.2.0	Jupiter integration
	Wallet Adapter	0.15.35	Wallet connection
3*State	Zustand	4.5.0	State management
	React Query	5.28.14	Server state
	React Hook Form	7.51.0	Form handling
3*UI/UX	Radix UI	Latest	Headless components
	Lucide React	Latest	Icons
	Framer Motion	11.0.0	Animations
3*Testing	Jest	29.7.0	Unit testing
	Testing Library	14.2.0	Component testing
	Playwright	1.42.0	E2E testing
2*DevOps	ESLint	8.57.0	Code linting
	Prettier	3.2.0	Code formatting

Table 1: Complete Technology Stack

3 Core Services

3.1 Jupiter Service Integration

```
/**
 * Jupiter API v6 Integration Service
 * Handles quote fetching, swap transactions, and route optimization
 */
export class JupiterService {
  private readonly apiBase: string;
  private readonly version: string;

  constructor() {
    this.apiBase = process.env.NEXT_PUBLIC_JUPITER_API_BASE!;
    this.version = process.env.NEXT_PUBLIC_JUPITER_API_VERSION!;
  }

  /**
   * Fetch optimized quote for token swap
   * @param params - Quote parameters
   * @returns Promise<JupiterQuote>
   */
  async getQuote(params: QuoteParams): Promise<JupiterQuote> {
    const queryParams = new URLSearchParams({
      inputMint: params.inputMint,
      outputMint: params.outputMint,
      amount: params.amount.toString(),
      slippageBps: params.slippageBps?.toString() || '50',
      feeBps: params.feeBps?.toString() || '0',
      onlyDirectRoutes: params.onlyDirectRoutes?.toString() || 'false',
      asLegacyTransaction: 'false',
      platformFeeBps: '25', // 0.25% platform fee
      maxAccounts: '64',
    });

    const response = await fetch(
      `${this.apiBase}/${this.version}/quote?${queryParams}`,
      {
        method: 'GET',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json',
        },
      }
    );

    if (!response.ok) {
      throw new JupiterApiError(
        `Quote request failed: ${response.status}`,
        response.status
      );
    }

    return response.json();
  }

  /**
   * Get swap transaction for execution
   * @param quoteResponse - Quote from getQuote()
   * @param userPublicKey - User's wallet public key
   * @param options - Additional swap options
   * @returns Promise<SwapTransaction>
   */
  async getSwapTransaction(
    quoteResponse: JupiterQuote,
    userPublicKey: PublicKey,
    options: SwapOptions = {}
  ): Promise<SwapTransaction> {
    const swapRequest = {
      quoteResponse,
      userPublicKey: userPublicKey.toString(),
      wrapAndUnwrapSol: true,
      useSharedAccounts: true,
      feeAccount: options.feeAccount,
      trackingAccount: options.trackingAccount,
      computeUnitPriceMicroLamports: options.priorityFee || 'auto',
      asLegacyTransaction: false,
      useTokenLedger: false,
      destinationTokenAccount: options.destinationTokenAccount,
    };

    const response = await fetch(`${this.apiBase}/${this.version}/swap`, {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(swapRequest),
    });

    if (!response.ok) {
      throw new JupiterApiError(
        `Swap transaction request failed: ${response.status}`,
        response.status
      );
    }

    return response.json();
  }
}
```

Listing 4: Jupiter API Service Implementation

3.2 Optimization Service

```
/**
 * Trading Optimization Service
 * Implements dynamic slippage, smart priority fees, and MEV protection
 */
export class OptimizationService {
  private readonly coingeckoService: CoingeckoService;
  private readonly rpcManager: RpcManager;

  constructor(
    coingeckoService: CoingeckoService,
    rpcManager: RpcManager
  ) {
    this.coingeckoService = coingeckoService;
    this.rpcManager = rpcManager;
  }

  /**
   * Calculate dynamic slippage based on market conditions
   * @param inputToken - Input token information
   * @param outputToken - Output token information
   * @param tradeSize - Trade size in USD
   * @returns Optimized slippage in basis points
   */
  async calculateDynamicSlippage(
    inputToken: Token,
    outputToken: Token,
    tradeSize: number
  ): Promise<number> {
    try {
      // Get market data for both tokens
      const [inputMarketData, outputMarketData] = await Promise.all([
        this.coingeckoService.getTokenMarketData(inputToken.coingeckoId),
        this.coingeckoService.getTokenMarketData(outputToken.coingeckoId),
      ]);

      // Base slippage (0.5%)
      const baseSlippage = 50;

      // Volatility factor (based on 24h price change)
      const inputVolatility = Math.abs(inputMarketData.price_change_percentage_24h ||
    0);
      const outputVolatility = Math.abs(outputMarketData.price_change_percentage_24h
   || 0);
      const avgVolatility = (inputVolatility + outputVolatility) / 2;

      const volatilityFactor = Math.min(2.0, Math.max(0.8, 1.0 + (avgVolatility / 100)));

      // Trade size factor (larger trades need more slippage)
      const sizeFactor = Math.min(1.5, 1.0 + Math.log10(tradeSize / 1000) * 0.1);

      // Liquidity factor (based on 24h volume)
      const avgVolume = (inputMarketData.total_volume + outputMarketData.total_volume) / 2;
      const liquidityFactor = Math.min(1.3, Math.max(0.7, 1.0 - Math.log10(avgVolume / 1000000) * 0.1));

      // Calculate dynamic slippage
      const dynamicSlippage = Math.round(
        baseSlippage * volatilityFactor * sizeFactor * liquidityFactor
      );

      // Ensure slippage is within reasonable bounds (0.1% to 3%)
      return Math.min(300, Math.max(10, dynamicSlippage));
    } catch (error) {
      console.warn('Failed to calculate dynamic slippage, using default:', error);
      return 50; // Default 0.5%
    }
  }

  /**
   * Calculate smart priority fee based on network conditions
   * @param urgency - Transaction urgency level
   * @returns Priority fee in microLamports
   */
  async calculateSmartPriorityFee(
    urgency: 'low' | 'medium' | 'high' = 'medium'
  ): Promise<number> {
    try {
      const connection = this.rpcManager.getConnection();

      // Get recent prioritization fees
      const recentFees = await connection.getRecentPrioritizationFees({
        lockedWritableAccounts: [
          new PublicKey('11111111111111111111111111111112'), // System Program
          new PublicKey('TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA'), // Token Program
        ],
      });

      if (recentFees.length === 0) {
        return this.getDefaultPriorityFee(urgency);
      }

      // Calculate percentiles
      const fees = recentFees.map(fee => fee.prioritizationFee).sort((a, b) => a - b);
      const p50 = fees[Math.floor(fees.length * 0.5)];
      const p75 = fees[Math.floor(fees.length * 0.75)];
      const p90 = fees[Math.floor(fees.length * 0.9)];

      // Select fee based on urgency
      let targetFee: number;
      switch (urgency) {
        case 'low':
          targetFee = p50;
          break;
        case 'medium':
          targetFee = p75;
          break;
        case 'high':
          targetFee = p90;
          break;
      }

      // Apply bounds and return
      return Math.min(100000, Math.max(1000, targetFee));
    } catch (error) {
      console.warn('Failed to calculate smart priority fee, using default:', error);
      return this.getDefaultPriorityFee(urgency);
    }
  }

  private getDefaultPriorityFee(urgency: 'low' | 'medium' | 'high'): number {
    const defaultFees = {
      low: 1000,     // 0.001 SOL
      medium: 5000, // 0.005 SOL
      high: 10000,  // 0.01 SOL
    };
    return defaultFees[urgency];
  }
}
```

Listing 5: Advanced Trading Optimizations

4 Security Error Handling

4.1 Comprehensive Error Management

```
/**
 * Custom Error Classes for Jupiter Swap DApp
 * Provides detailed error information for better debugging and user experience
 */

export class JupiterSwapError extends Error {
  public readonly code: string;
  public readonly context?: Record<string, any>;
  public readonly timestamp: Date;

  constructor(message: string, code: string, context?: Record<string, any>) {
    super(message);
    this.name = 'JupiterSwapError';
    this.code = code;
    this.context = context;
    this.timestamp = new Date();
  }
}

export class WalletError extends JupiterSwapError {
  constructor(message: string, context?: Record<string, any>) {
    super(message, 'WALLET_ERROR', context);
    this.name = 'WalletError';
  }
}

export class TransactionError extends JupiterSwapError {
  public readonly signature?: string;

  constructor(message: string, signature?: string, context?: Record<string, any>) {
    super(message, 'TRANSACTION_ERROR', context);
    this.name = 'TransactionError';
    this.signature = signature;
  }
}

export class JupiterApiError extends JupiterSwapError {
  public readonly statusCode?: number;

  constructor(message: string, statusCode?: number, context?: Record<string, any>) {
    super(message, 'JUPITER_API_ERROR', context);
    this.name = 'JupiterApiError';
    this.statusCode = statusCode;
  }
}

export class RpcError extends JupiterSwapError {
  public readonly endpoint?: string;

  constructor(message: string, endpoint?: string, context?: Record<string, any>) {
    super(message, 'RPC_ERROR', context);
    this.name = 'RpcError';
    this.endpoint = endpoint;
  }
}

/**
 * Error Handler Utility
 * Centralized error processing and logging
 */
export class ErrorHandler {
  static handle(error: unknown, context?: string): JupiterSwapError {
    // Log error to console and Sentry
    console.error(`[${context || 'Unknown'}] Error:`, error);

    if (typeof window !== 'undefined' && window.Sentry) {
      window.Sentry.captureException(error, {
        tags: { context },
        extra: { timestamp: new Date().toISOString() }
      });
    }

    // Convert to typed error
    if (error instanceof JupiterSwapError) {
      return error;
    }

    if (error instanceof Error) {
      return new JupiterSwapError(
        error.message,
        'UNKNOWN_ERROR',
        { originalError: error.name, context }
      );
    }

    return new JupiterSwapError(
      'An unknown error occurred',
      'UNKNOWN_ERROR',
      { originalError: String(error), context }
    );
  }
}
```

Listing 6: Typed Error System

5 Testing Strategy

5.1 Test Configuration

```
// jest.config.js
module.exports = {
  preset: 'ts-jest',
  testEnvironment: 'jsdom',
  roots: ['<rootDir>/src'],
  transform: {
    '^.+\\.tsx?$': ['ts-jest', {
      tsconfig: 'tsconfig.test.json',
    }],
  },
  moduleNameMapper: {
    '^@/(.*)$': '<rootDir>/src/$1',
    '^@/components/(.*)$': '<rootDir>/src/components/$1',
    '^@/services/(.*)$': '<rootDir>/src/services/$1',
    '^@/hooks/(.*)$': '<rootDir>/src/hooks/$1',
    '^@/store/(.*)$': '<rootDir>/src/store/$1',
    '^@/types/(.*)$': '<rootDir>/src/types/$1',
    '^@/utils/(.*)$': '<rootDir>/src/utils/$1',
    '^@/constants$': '<rootDir>/src/constants/index.ts',
  },
  setupFilesAfterEnv: ['<rootDir>/src/__tests__/setup.ts'],
  collectCoverageFrom: [
    'src/**/*.{ts,tsx}',
    '!src/**/*.d.ts',
    '!src/**/__mocks__/**',
    '!src/**/__tests__/**',
  ],
  coverageThreshold: {
    global: {
      branches: 70,
      functions: 70,
      lines: 70,
      statements: 70,
    },
  },
};
```

Listing 7: Jest Configuration with Mocks

5.2 Test Examples

```
// src/services/__tests__/swap.test.ts
import { SwapService } from '../swap';
import { JupiterService } from '../jupiter';
import { OptimizationService } from '../optimization';

describe('SwapService', () => {
  let swapService: SwapService;
  let mockJupiterService: jest.Mocked<JupiterService>;
  let mockOptimizationService: jest.Mocked<OptimizationService>;

  beforeEach(() => {
    mockJupiterService = {
      getQuote: jest.fn(),
      getSwapTransaction: jest.fn(),
    } as any;

    mockOptimizationService = {
      calculateDynamicSlippage: jest.fn(),
      calculateSmartPriorityFee: jest.fn(),
    } as any;

    swapService = new SwapService(
      mockJupiterService,
      mockOptimizationService
    );
  });

  describe('executeSwap', () => {
    it('should execute swap with optimizations', async () => {
      // Mock responses
      mockOptimizationService.calculateDynamicSlippage.mockResolvedValue(75);
      mockOptimizationService.calculateSmartPriorityFee.mockResolvedValue(5000);

      mockJupiterService.getQuote.mockResolvedValue({
        inputMint: 'So11111111111111111111111111111111111111112',
        outputMint: 'EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v',
        inAmount: '1000000000',
        outAmount: '180500000',
        slippageBps: 75,
        routePlan: [],
      });

      mockJupiterService.getSwapTransaction.mockResolvedValue({
        swapTransaction: 'base64_transaction_data',
        lastValidBlockHeight: 123456789,
      });

      const result = await swapService.executeSwap({
        inputToken: SOL_TOKEN,
        outputToken: USDC_TOKEN,
        inputAmount: 1.0,
        userPublicKey: new PublicKey('11111111111111111111111111111112'),
        enableOptimizations: true,
      });

      expect(result.success).toBe(true);
      expect(mockOptimizationService.calculateDynamicSlippage).toHaveBeenCalled();
      expect(mockOptimizationService.calculateSmartPriorityFee).toHaveBeenCalled();
    });

    it('should handle swap errors gracefully', async () => {
      mockJupiterService.getQuote.mockRejectedValue(
        new Error('Insufficient liquidity')
      );

      await expect(
        swapService.executeSwap({
          inputToken: SOL_TOKEN,
          outputToken: USDC_TOKEN,
          inputAmount: 1000000, // Unrealistic amount
          userPublicKey: new PublicKey('11111111111111111111111111111112'),
        })
      ).rejects.toThrow('Insufficient liquidity');
    });
  });
});
```

Listing 8: Service Unit Tests

6 Performance Optimization

6.1 Bundle Optimization

```
// next.config.js
/** @type {import('next').NextConfig} */
const nextConfig = {
  // Enable experimental features
  experimental: {
    optimizePackageImports: ['@radix-ui/react-icons', 'lucide-react'],
    turbo: {
      rules: {
        '*.svg': {
          loaders: ['@svgr/webpack'],
          as: '*.js',
        },
      },
    },
  },

  // Webpack optimizations
  webpack: (config, { dev, isServer }) => {
    // Optimize bundle splitting
    if (!dev && !isServer) {
      config.optimization.splitChunks = {
        chunks: 'all',
        cacheGroups: {
          solana: {
            name: 'solana-vendors',
            test: /[\\/]node_modules[\\/](@solana|@jup-ag)[\\/]/,
            priority: 10,
            reuseExistingChunk: true,
          },
          ui: {
            name: 'ui-vendors',
            test: /[\\/]node_modules[\\/](@radix-ui|lucide-react)[\\/]/,
            priority: 9,
            reuseExistingChunk: true,
          },
          default: {
            minChunks: 2,
            priority: -10,
            reuseExistingChunk: true,
          },
        },
      };
    }

    // Resolve fallbacks for Node.js modules
    config.resolve.fallback = {
      ...config.resolve.fallback,
      crypto: require.resolve('crypto-browserify'),
      stream: require.resolve('stream-browserify'),
      buffer: require.resolve('buffer'),
    };

    return config;
  },

  // Image optimization
  images: {
    domains: ['raw.githubusercontent.com'],
    formats: ['image/webp', 'image/avif'],
  },

  // Security headers
  async headers() {
    return [
      {
        source: '/(.*)',
        headers: [
          {
            key: 'X-Frame-Options',
            value: 'DENY',
          },
          {
            key: 'X-Content-Type-Options',
            value: 'nosniff',
          },
          {
            key: 'Referrer-Policy',
            value: 'strict-origin-when-cross-origin',
          },
          {
            key: 'Content-Security-Policy',
            value: "default-src 'self'; script-src 'self' 'unsafe-eval' 'unsafe-
   inline'; style-src 'self' 'unsafe-inline'; img-src 'self' data: https:; font-src
   'self' data:; connect-src 'self' https:",
          },
        ],
      },
    ];
  },
};

module.exports = nextConfig;
```

Listing 9: Next.js Configuration for Performance

7 Deployment Guide

7.1 Production Deployment

```
# Build for production
npm run build

# Test production build locally
npm run start

# Deploy to Vercel
vercel --prod

# Or deploy with GitHub Actions
git push origin main
```

Listing 10: Production Build and Deployment

7.2 Environment-Specific Configurations

Environment	Network	RPC	Features
Development	Devnet	Local/Devnet	All features + devtools
Staging	Devnet	Helius Devnet	Production simulation
Production	Mainnet	Helius + Alchemy	Full production

Table 2: Environment Configurations

8 API Reference

8.1 Core Hooks

```
/**
 * useSwap Hook - Main swap functionality
 */
export function useSwap() {
  const {
    inputToken,
    outputToken,
    inputAmount,
    outputAmount,
    quote,
    isLoading,
    error,
    setInputToken,
    setOutputToken,
    setInputAmount,
    fetchQuote,
    executeSwap,
    reset,
  } = useSwapStore();

  const { publicKey, connected } = useWallet();

  // Auto-fetch quote when parameters change
  useEffect(() => {
    if (inputToken && outputToken && inputAmount && connected && publicKey) {
      const debounceTimer = setTimeout(() => {
        fetchQuote(publicKey);
      }, 500);

      return () => clearTimeout(debounceTimer);
    }
  }, [inputToken, outputToken, inputAmount, connected, publicKey]);

  return {
    // State
    inputToken,
    outputToken,
    inputAmount,
    outputAmount,
    quote,
    isLoading,
    error,
    canSwap: connected && inputToken && outputToken && inputAmount && quote,

    // Actions
    setInputToken,
    setOutputToken,
    setInputAmount,
    executeSwap: () => executeSwap(publicKey!),
    reset,
  };
}

/**
 * useOptimization Hook - Trading optimizations
 */
export function useOptimization() {
  const {
    optimizationsEnabled,
    dynamicSlippage,
    smartPriorityFee,
    mevProtection,
    toggleOptimizations,
    updateSlippage,
    updatePriorityFee,
  } = useOptimizationStore();

  return {
    // State
    optimizationsEnabled,
    dynamicSlippage,
    smartPriorityFee,
    mevProtection,

    // Actions
    toggleOptimizations,
    updateSlippage,
    updatePriorityFee,
  };
}
```

Listing 11: Custom React Hooks

9 Troubleshooting

9.1 Common Issues

Issue: "Transaction signature verification failure"

- **Cause:** RPC endpoint issues or network congestion
- **Solution:** Check RPC configuration, try different endpoint
- **Code:** Verify `NEXT_PUBLIC_HELIUS_API_KEY` is valid

Issue: "Insufficient SOL for transaction"

- **Cause:** Not enough SOL for transaction fees
- **Solution:** Ensure wallet has at least 0.01 SOL for fees
- **Prevention:** Implement balance checks before swap

Issue: "Slippage tolerance exceeded"

- **Cause:** High market volatility or large trade size
- **Solution:** Increase slippage tolerance or reduce trade size
- **Feature:** Dynamic slippage optimization helps prevent this

9.2 Debug Commands

```
# Check build issues
npm run build 2>&1 | tee build.log

# Run tests with coverage
npm run test:coverage

# Lint and fix issues
npm run lint:fix

# Type checking
npm run type-check

# Analyze bundle size
npm run analyze

# Check for security vulnerabilities
npm audit

# Update dependencies
npm update --save
```

Listing 12: Debugging Commands

10 Contributing

10.1 Development Workflow

```
# 1. Fork and clone the repository
git clone https://github.com/your-username/jupiter-swap-nextjs.git
cd jupiter-swap-nextjs

# 2. Create a feature branch
git checkout -b feature/your-feature-name

# 3. Install dependencies
npm install

# 4. Make your changes
# ... code changes ...

# 5. Run tests
npm run test

# 6. Run linting
npm run lint:fix

# 7. Commit changes
git add .
git commit -m "feat: add your feature description"
```

```
# 8. Push to your fork
git push origin feature/your-feature-name

# 9. Create a Pull Request
# Open GitHub and create a PR from your fork
```

Listing 13: Development Workflow

10.2 Code Standards

- **TypeScript:** Strict mode enabled, no `any` types
- **ESLint:** Airbnb configuration with custom rules
- **Prettier:** Automatic code formatting
- **Husky:** Pre-commit hooks for quality checks
- **Conventional Commits:** Standardized commit messages

11 License & Credits

11.1 License

This project is licensed under the MIT License. See the `LICENSE` file for details.

11.2 Credits

- **Developer:** Kamel (@treizeb__)
- **Company:** DeAura.io
- **Jupiter Protocol:** Jupiter Exchange
- **Solana Foundation:** Solana Blockchain
- **Vercel:** Deployment Platform

11.3 Acknowledgments

Special thanks to the Jupiter team for their excellent API documentation and the Solana community for their continuous support and innovation in the DeFi space.

Built with for the Solana ecosystem
DeAura.io - July 2025