Complete Project Architecture

Jupiter Swap DApp

Technical Guide

Comprehensive Architecture Documentation

Frontend: Next.js 14 + TypeScript
Blockchain: Solana + Jupiter API v6
State Management: Zustand + React Query
UI Framework: Tailwind + shadcn/ui
Testing: Jest + Testing Library
Deployment: Vercel + GitHub Actions
Monitoring: Sentry + Custom Analytics
Security: Multi-layer Protection

Documentation Coverage

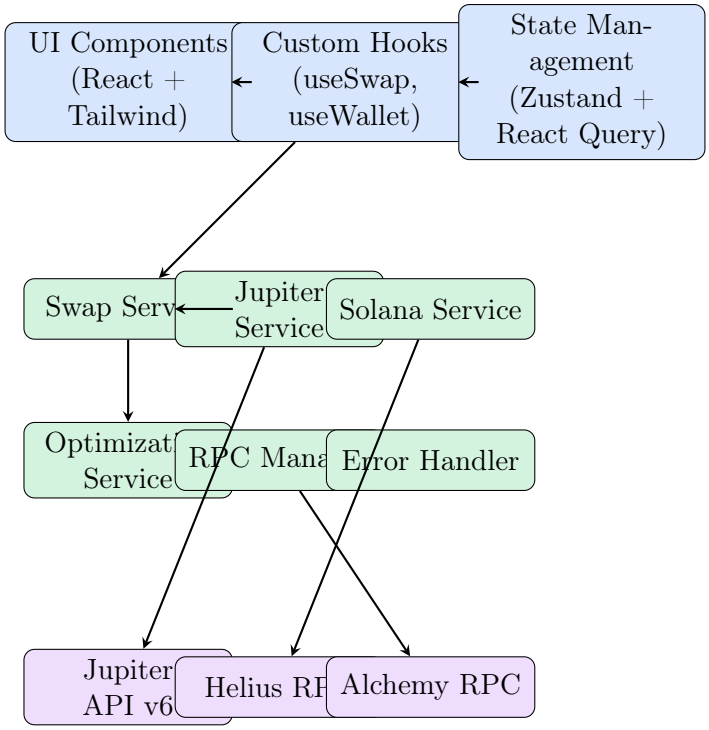
Component Hierarchy & Design Patterns
Service Layer Architecture
State Management Strategy
API Integration Patterns
Security Implementation
Performance Optimization
Testing Architecture
Deployment Pipeline

Author: Kamel (@treizeb__)
Company: DeAura.io
Updated: July 14, 2025

Contents

1 Architecture Overview

1.1 System Architecture Diagram



1.2 Architecture Principles

Core Principles:

1. **Separation of Concerns:** Clear boundaries between UI, business logic, and data layers
2. **Modularity:** Loosely coupled, highly cohesive components
3. **Scalability:** Designed to handle increased load and feature expansion
4. **Maintainability:** Clean code with comprehensive documentation
5. **Testability:** Architecture supports comprehensive testing strategies
6. **Performance:** Optimized for speed and efficiency
7. **Security:** Multi-layer security implementation
8. **Reliability:** Robust error handling and failover mechanisms

2 Component Hierarchy

2.1 Frontend Component Structure

```
src/components/
          providers/
                Providers.tsx                # Root providers wrapper
                WalletProvider.tsx           # Solana wallet integration
                QueryProvider.tsx           # React Query setup
```

```
            ThemeProvider.tsx          # Theme management
      layout/
            Header.tsx                  # Application header
            Footer.tsx                  # Application footer
            Navigation.tsx              # Main navigation
            Sidebar.tsx                 # Optional sidebar
      swap/
            SwapInterface.tsx           # Main swap component
            TokenSelector.tsx           # Token selection dropdown
            AmountInput.tsx             # Amount input with validation
            SwapButton.tsx              # Swap execution button
            QuoteDisplay.tsx            # Quote information display
            SlippageSettings.tsx        # Slippage configuration
            TransactionStatus.tsx       # Transaction progress
      wallet/
            WalletConnectButton.tsx     # Wallet connection
            WalletInfo.tsx              # Wallet information display
            BalanceDisplay.tsx          # Token balances
            DisconnectButton.tsx        # Wallet disconnection
      analytics/
            NetworkStatus.tsx           # Network health indicator
            OptimizationPanel.tsx       # Optimization settings
            TransactionHistory.tsx      # Transaction history
            PerformanceMetrics.tsx      # Performance dashboard
            FeeAnalytics.tsx            # Fee analysis
      ui/                              # shadcn/ui components
            button.tsx                  # Button component
            card.tsx                    # Card component
            input.tsx                   # Input component
            dialog.tsx                  # Dialog/Modal component
            toast.tsx                   # Toast notifications
            progress.tsx                # Progress indicators
            tabs.tsx                    # Tab component
            tooltip.tsx                 # Tooltip component
            select.tsx                  # Select dropdown
            badge.tsx                   # Badge component
            separator.tsx               # Separator component
            scroll-area.tsx             # Scroll area
      errors/
          ErrorBoundary.tsx            # React error boundary
          ErrorDisplay.tsx             # Error message display
          FallbackComponent.tsx        # Fallback UI component
```

Listing 1: Component Hierarchy Tree

2.2 Component Design Patterns

2.2.1 Compound Component Pattern

```tsx
// SwapInterface.tsx - Main compound component
export function SwapInterface() {
  return (
    <SwapProvider>
      <Card className="w-full max-w-md mx-auto">
        <SwapInterface.Header />
        <SwapInterface.Body>
          <SwapInterface.TokenInput type="input" />
          <SwapInterface.SwapButton />
          <SwapInterface.TokenInput type="output" />
        </SwapInterface.Body>
        <SwapInterface.Footer />
      </Card>
```

```
    </SwapProvider>
  );
}

// Compound component parts
SwapInterface.Header = function SwapHeader() {
  return (
    <CardHeader>
      <CardTitle>Swap Tokens</CardTitle>
      <CardDescription>
        Exchange SOL and USDC with optimized rates
      </CardDescription>
    </CardHeader>
  );
};

SwapInterface.Body = function SwapBody({ children }: { children: React.ReactNode }) {
  return (
    <CardContent className="space-y-4">
      {children}
    </CardContent>
  );
};

SwapInterface.TokenInput = function TokenInput({ type }: { type: 'input' | 'output' }) {
  const { inputToken, outputToken, inputAmount, outputAmount } = useSwap();

  return (
    <div className="space-y-2">
      <Label>{type === 'input' ? 'From' : 'To'}</Label>
      <div className="flex space-x-2">
        <TokenSelector
          selectedToken={type === 'input' ? inputToken : outputToken}
          onSelectToken={type === 'input' ? setInputToken : setOutputToken}
        />
        <AmountInput
          value={type === 'input' ? inputAmount : outputAmount}
          onChange={type === 'input' ? setInputAmount : undefined}
          readOnly={type === 'output'}
        />
      </div>
    </div>
  );
};

SwapInterface.SwapButton = function SwapButton() {
  const { canSwap, isLoading, executeSwap } = useSwap();

  return (
    <Button
      onClick={executeSwap}
      disabled={!canSwap || isLoading}
      className="w-full"
      size="lg"
    >
      {isLoading ? (
        <>
          <Loader2 className="mr-2 h-4 w-4 animate-spin" />
          Processing...
        </>
      ) : (
        'Swap Tokens'
```

```
      )}
    </Button>
  );
};

SwapInterface.Footer = function SwapFooter() {
  const { quote, optimizationsEnabled } = useSwap();

  if (!quote) return null;

  return (
    <CardFooter>
      <div className="w-full space-y-2 text-sm text-muted-foreground">
        <div className="flex justify-between">
          <span>Rate</span>
          <span>{quote.rate}</span>
        </div>
        <div className="flex justify-between">
          <span>Optimizations</span>
          <span>{optimizationsEnabled ? 'Enabled' : 'Disabled'}</span>
        </div>
      </div>
    </CardFooter>
  );
};
```

Listing 2: SwapInterface Compound Component

2.2.2 Hook-Based State Management Pattern

```
// hooks/useSwap.ts - Main swap hook
export function useSwap() {
  const store = useSwapStore();
  const { publicKey, connected } = useWallet();
  const { connection } = useConnection();

  // Derived state
  const canSwap = useMemo(() => {
    return connected &&
           store.inputToken &&
           store.outputToken &&
           store.inputAmount &&
           store.quote &&
           !store.isLoading;
  }, [connected, store.inputToken, store.outputToken, store.inputAmount, store.quote,
     store.isLoading]);

  // Auto-fetch quote effect
  useEffect(() => {
    if (store.inputToken && store.outputToken && store.inputAmount && connected &&
    publicKey) {
      const timeoutId = setTimeout(() => {
        store.fetchQuote(publicKey, connection);
      }, 500); // Debounce

      return () => clearTimeout(timeoutId);
    }
  }, [store.inputToken, store.outputToken, store.inputAmount, connected, publicKey,
    connection]);

  // Cleanup effect
  useEffect(() => {
```

```
    return () => {
      store.reset();
    };
  }, []);

  return {
    // State
    ...store,
    canSwap,

    // Actions with context
    executeSwap: useCallback(() => {
      if (publicKey && connection) {
        return store.executeSwap(publicKey, connection);
      }
      throw new Error('Wallet not connected');
    }, [publicKey, connection, store.executeSwap]),
  };
}

// hooks/useOptimization.ts - Optimization hook
export function useOptimization() {
  const store = useOptimizationStore();
  const { inputToken, outputToken, inputAmount } = useSwap();

  // Auto-calculate optimizations
  useEffect(() => {
    if (store.optimizationsEnabled && inputToken && outputToken && inputAmount) {
      store.calculateOptimizations(inputToken, outputToken, parseFloat(inputAmount));
    }
  }, [store.optimizationsEnabled, inputToken, outputToken, inputAmount]);

  return {
    ...store,

    // Computed values
    estimatedSavings: useMemo(() => {
      if (!store.dynamicSlippage || !store.smartPriorityFee) return 0;
      return store.slippageSavings + store.priorityFeeSavings;
    }, [store.dynamicSlippage, store.smartPriorityFee, store.slippageSavings, store.
    priorityFeeSavings]),
  };
}
```

Listing 3: Custom Hook Pattern Implementation

3 Service Layer Design

3.1 Service Architecture Pattern

```
// services/base/BaseService.ts - Abstract base service
export abstract class BaseService {
  protected readonly logger: Logger;
  protected readonly errorHandler: ErrorHandler;

  constructor(logger: Logger, errorHandler: ErrorHandler) {
    this.logger = logger;
    this.errorHandler = errorHandler;
  }

  protected async executeWithRetry<T>(
```

```
    operation: () => Promise<T>,
    maxRetries: number = 3,
    delay: number = 1000
  ): Promise<T> {
    let lastError: Error;

    for (let attempt = 1; attempt <= maxRetries; attempt++) {
      try {
        return await operation();
      } catch (error) {
        lastError = error as Error;
        this.logger.warn(`Attempt ${attempt} failed:`, error);

        if (attempt < maxRetries) {
          await new Promise(resolve => setTimeout(resolve, delay * attempt));
        }
      }
    }

    throw this.errorHandler.handle(lastError!, `Failed after ${maxRetries} attempts`);
  }

  protected validateRequired<T>(value: T | null | undefined, fieldName: string): T {
    if (value === null || value === undefined) {
      throw new ValidationError(`${fieldName} is required`);
    }
    return value;
  }
}

// services/jupiter.ts - Jupiter service implementation
export class JupiterService extends BaseService {
  private readonly apiBase: string;
  private readonly version: string;
  private readonly httpClient: HttpClient;

  constructor(
    logger: Logger,
    errorHandler: ErrorHandler,
    httpClient: HttpClient
  ) {
    super(logger, errorHandler);
    this.apiBase = this.validateRequired(
      process.env.NEXT_PUBLIC_JUPITER_API_BASE,
      'NEXT_PUBLIC_JUPITER_API_BASE'
    );
    this.version = process.env.NEXT_PUBLIC_JUPITER_API_VERSION || 'v6';
    this.httpClient = httpClient;
  }

  async getQuote(params: QuoteParams): Promise<JupiterQuote> {
    this.logger.info('Fetching Jupiter quote', { params });

    return this.executeWithRetry(async () => {
      const queryParams = this.buildQuoteParams(params);
      const response = await this.httpClient.get(
        `${this.apiBase}/${this.version}/quote`,
        { params: queryParams }
      );

      return this.validateQuoteResponse(response.data);
    });
```

```
  }

  async getSwapTransaction(
    quote: JupiterQuote,
    userPublicKey: PublicKey,
    options: SwapOptions = {}
  ): Promise<SwapTransaction> {
    this.logger.info('Getting swap transaction', { quote, userPublicKey:
     userPublicKey.toString() });

    return this.executeWithRetry(async () => {
      const swapRequest = this.buildSwapRequest(quote, userPublicKey, options);
      const response = await this.httpClient.post(
        `${this.apiBase}/${this.version}/swap`,
        swapRequest
      );

      return this.validateSwapResponse(response.data);
    });
  }

  private buildQuoteParams(params: QuoteParams): Record<string, string> {
    return {
      inputMint: params.inputMint,
      outputMint: params.outputMint,
      amount: params.amount.toString(),
      slippageBps: (params.slippageBps || 50).toString(),
      feeBps: (params.feeBps || 0).toString(),
      onlyDirectRoutes: (params.onlyDirectRoutes || false).toString(),
      asLegacyTransaction: 'false',
      platformFeeBps: '25',
      maxAccounts: '64',
    };
  }

  private validateQuoteResponse(data: any): JupiterQuote {
    if (!data.inputMint || !data.outputMint || !data.inAmount || !data.outAmount) {
      throw new ValidationError('Invalid quote response structure');
    }
    return data as JupiterQuote;
  }

  private validateSwapResponse(data: any): SwapTransaction {
    if (!data.swapTransaction) {
      throw new ValidationError('Invalid swap transaction response');
    }
    return data as SwapTransaction;
  }
}
```

Listing 4: Service Layer Architecture

3.2 Service Dependency Injection

```
// services/container.ts - Service container
export class ServiceContainer {
  private services = new Map<string, any>();
  private factories = new Map<string, () => any>();

  register<T>(name: string, factory: () => T): void {
    this.factories.set(name, factory);
  }
```

```
9
10    get<T>(name: string): T {
11      if (this.services.has(name)) {
12        return this.services.get(name);
13      }
14
15      const factory = this.factories.get(name);
16      if (!factory) {
17        throw new Error(`Service ${name} not registered`);
18      }
19
20      const service = factory();
21      this.services.set(name, service);
22      return service;
23    }
24
25    singleton<T>(name: string, instance: T): void {
26      this.services.set(name, instance);
27    }
28  }
29
30  // services/registry.ts - Service registration
31  export function createServiceContainer(): ServiceContainer {
32    const container = new ServiceContainer();
33
34    // Core services
35    container.register('logger', () => new Logger());
36    container.register('errorHandler', () => new ErrorHandler(container.get('logger')))
        ;
37    container.register('httpClient', () => new HttpClient());
38
39    // RPC services
40    container.register('rpcManager', () => new RpcManager(
41      container.get('logger'),
42      container.get('errorHandler')
43    ));
44
45    // Blockchain services
46    container.register('jupiterService', () => new JupiterService(
47      container.get('logger'),
48      container.get('errorHandler'),
49      container.get('httpClient')
50    ));
51
52    container.register('solanaService', () => new SolanaService(
53      container.get('logger'),
54      container.get('errorHandler'),
55      container.get('rpcManager')
56    ));
57
58    // Business logic services
59    container.register('optimizationService', () => new OptimizationService(
60      container.get('logger'),
61      container.get('errorHandler'),
62      container.get('coingeckoService')
63    ));
64
65    container.register('swapService', () => new SwapService(
66      container.get('logger'),
67      container.get('errorHandler'),
68      container.get('jupiterService'),
69      container.get('solanaService'),
70      container.get('optimizationService')
```

```
  ));

  return container;
}

// hooks/useServices.ts - Service access hook
const ServiceContext = createContext<ServiceContainer | null>(null);

export function ServiceProvider({ children }: { children: React.ReactNode }) {
  const [container] = useState(() => createServiceContainer());

  return (
    <ServiceContext.Provider value={container}>
      {children}
    </ServiceContext.Provider>
  );
}

export function useService<T>(name: string): T {
  const container = useContext(ServiceContext);
  if (!container) {
    throw new Error('useService must be used within ServiceProvider');
  }
  return container.get<T>(name);
}

// Usage in components
export function SwapInterface() {
  const swapService = useService<SwapService>('swapService');
  const optimizationService = useService<OptimizationService>('optimizationService');

  // Component logic using services
}
```

Listing 5: Service Container and Dependency Injection

4 State Management Strategy

4.1 Zustand Store Architecture

```
// store/swapStore.ts - Main swap store
interface SwapState {
  // Token selection
  inputToken: Token | null;
  outputToken: Token | null;

  // Amounts
  inputAmount: string;
  outputAmount: string;

  // Quote data
  quote: JupiterQuote | null;
  quoteError: string | null;
  quoteLoading: boolean;

  // Swap execution
  swapStatus: 'idle' | 'pending' | 'success' | 'error';
  swapError: string | null;
  swapResult: SwapResult | null;

  // Transaction tracking
```

```
  currentTransaction: string | null;
  transactionHistory: SwapTransaction[];

  // Settings
  slippageTolerance: number;
  priorityFee: number;
  optimizationsEnabled: boolean;
}

interface SwapActions {
  // Token actions
  setInputToken: (token: Token | null) => void;
  setOutputToken: (token: Token | null) => void;
  swapTokens: () => void;

  // Amount actions
  setInputAmount: (amount: string) => void;
  setMaxAmount: () => void;

  // Quote actions
  fetchQuote: (userPublicKey: PublicKey, connection: Connection) => Promise<void>;
  clearQuote: () => void;

  // Swap actions
  executeSwap: (userPublicKey: PublicKey, connection: Connection) => Promise<
    SwapResult>;

  // Settings actions
  updateSlippage: (slippage: number) => void;
  updatePriorityFee: (fee: number) => void;
  toggleOptimizations: () => void;

  // Utility actions
  reset: () => void;
  addToHistory: (transaction: SwapTransaction) => void;
}

export const useSwapStore = create<SwapState & SwapActions>()(
  devtools(
    persist(
      (set, get) => ({
        // Initial state
        inputToken: DEFAULT_SOL_TOKEN,
        outputToken: DEFAULT_USDC_TOKEN,
        inputAmount: '',
        outputAmount: '',
        quote: null,
        quoteError: null,
        quoteLoading: false,
        swapStatus: 'idle',
        swapError: null,
        swapResult: null,
        currentTransaction: null,
        transactionHistory: [],
        slippageTolerance: 0.5,
        priorityFee: 0.005,
        optimizationsEnabled: true,

        // Token actions
        setInputToken: (token) => {
          set({ inputToken: token });
          if (token && get().outputToken) {
            get().fetchQuote();
```

```
        }
      },

      setOutputToken: (token) => {
        set({ outputToken: token });
        if (token && get().inputToken) {
          get().fetchQuote();
        }
      },

      swapTokens: () => {
        const { inputToken, outputToken, inputAmount, outputAmount } = get();
        set({
          inputToken: outputToken,
          outputToken: inputToken,
          inputAmount: outputAmount,
          outputAmount: inputAmount,
        });
      },

      // Amount actions
      setInputAmount: (amount) => {
        set({ inputAmount: amount });
        if (amount && get().inputToken && get().outputToken) {
          get().fetchQuote();
        }
      },

      setMaxAmount: async () => {
        const { inputToken } = get();
        if (!inputToken) return;

        try {
          const balance = await getTokenBalance(inputToken);
          set({ inputAmount: balance.toString() });
          get().fetchQuote();
        } catch (error) {
          console.error('Failed to get max amount:', error);
        }
      },

      // Quote actions
      fetchQuote: async (userPublicKey, connection) => {
        const { inputToken, outputToken, inputAmount, optimizationsEnabled } = get
    ();

        if (!inputToken || !outputToken || !inputAmount || !userPublicKey) {
          return;
        }

        set({ quoteLoading: true, quoteError: null });

        try {
          const container = getServiceContainer();
          const swapService = container.get<SwapService>('swapService');

          const quote = await swapService.getOptimizedQuote({
            inputToken,
            outputToken,
            inputAmount: parseFloat(inputAmount),
            userPublicKey,
            enableOptimizations: optimizationsEnabled,
          });
```

```
146
147             set({
148               quote,
149               outputAmount: formatTokenAmount(quote.outAmount, outputToken.decimals),
150               quoteLoading: false,
151             });
152           } catch (error) {
153             set({
154               quoteError: error instanceof Error ? error.message : 'Failed to fetch
    quote',
155               quoteLoading: false,
156             });
157           }
158         },
159
160         // Swap actions
161         executeSwap: async (userPublicKey, connection) => {
162           const { quote, inputToken, outputToken } = get();
163
164           if (!quote || !inputToken || !outputToken) {
165             throw new Error('Missing required data for swap');
166           }
167
168           set({ swapStatus: 'pending', swapError: null });
169
170           try {
171             const container = getServiceContainer();
172             const swapService = container.get<SwapService>('swapService');
173
174             const result = await swapService.executeSwap({
175               quote,
176               userPublicKey,
177               connection,
178               onProgress: (stage, progress) => {
179                 // Update UI with progress
180                 console.log(`Swap ${stage}: ${progress}%`);
181               },
182             });
183
184             set({
185               swapStatus: 'success',
186               swapResult: result,
187               currentTransaction: result.signature,
188             });
189
190             // Add to history
191             get().addToHistory({
192               signature: result.signature,
193               inputToken,
194               outputToken,
195               inputAmount: get().inputAmount,
196               outputAmount: get().outputAmount,
197               timestamp: new Date(),
198             });
199
200             return result;
201           } catch (error) {
202             set({
203               swapStatus: 'error',
204               swapError: error instanceof Error ? error.message : 'Swap failed',
205             });
206             throw error;
207           }
```

```
        },

        // Utility actions
        reset: () => {
          set({
            inputAmount: '',
            outputAmount: '',
            quote: null,
            quoteError: null,
            quoteLoading: false,
            swapStatus: 'idle',
            swapError: null,
            swapResult: null,
            currentTransaction: null,
          });
        },

        addToHistory: (transaction) => {
          set((state) => ({
            transactionHistory: [transaction, ...state.transactionHistory].slice(0,
    50),
          }));
        },
      }),
      {
        name: 'jupiter-swap-store',
        partialize: (state) => ({
          inputToken: state.inputToken,
          outputToken: state.outputToken,
          slippageTolerance: state.slippageTolerance,
          priorityFee: state.priorityFee,
          optimizationsEnabled: state.optimizationsEnabled,
          transactionHistory: state.transactionHistory,
        }),
      }
    ),
    { name: 'SwapStore' }
  )
);
```

Listing 6: Zustand Store Implementation

5 API Integration Patterns

5.1 HTTP Client Architecture

```
// utils/httpClient.ts - HTTP client implementation
export class HttpClient {
  private readonly axiosInstance: AxiosInstance;
  private readonly logger: Logger;

  constructor(logger: Logger) {
    this.logger = logger;
    this.axiosInstance = axios.create({
      timeout: 30000,
      headers: {
        'Content-Type': 'application/json',
        'Accept': 'application/json',
      },
    });

```

```
    this.setupInterceptors();
  }

  private setupInterceptors(): void {
    // Request interceptor
    this.axiosInstance.interceptors.request.use(
      (config) => {
        const requestId = generateRequestId();
        config.metadata = { requestId, startTime: Date.now() };

        this.logger.debug('HTTP Request', {
          requestId,
          method: config.method?.toUpperCase(),
          url: config.url,
          params: config.params,
        });

        return config;
      },
      (error) => {
        this.logger.error('HTTP Request Error', error);
        return Promise.reject(error);
      }
    );

    // Response interceptor
    this.axiosInstance.interceptors.response.use(
      (response) => {
        const { requestId, startTime } = response.config.metadata || {};
        const duration = Date.now() - (startTime || 0);

        this.logger.debug('HTTP Response', {
          requestId,
          status: response.status,
          duration: `${duration}ms`,
          url: response.config.url,
        });

        return response;
      },
      (error) => {
        const { requestId, startTime } = error.config?.metadata || {};
        const duration = Date.now() - (startTime || 0);

        this.logger.error('HTTP Response Error', {
          requestId,
          status: error.response?.status,
          duration: `${duration}ms`,
          url: error.config?.url,
          message: error.message,
        });

        return Promise.reject(this.transformError(error));
      }
    );
  }

  private transformError(error: AxiosError): HttpError {
    if (error.response) {
      // Server responded with error status
      return new HttpError(
        error.response.data?.message || error.message,
        error.response.status,
```

```
        error.response.data
      );
    } else if (error.request) {
      // Request was made but no response received
      return new HttpError(
        'Network error - no response received',
        0,
        { originalError: error.message }
      );
    } else {
      // Request setup error
      return new HttpError(
        error.message,
        0,
        { originalError: error.message }
      );
    }
  }

  async get<T>(url: string, config?: AxiosRequestConfig): Promise<AxiosResponse<T>> {
    return this.axiosInstance.get<T>(url, config);
  }

  async post<T>(url: string, data?: any, config?: AxiosRequestConfig): Promise<
    AxiosResponse<T>> {
    return this.axiosInstance.post<T>(url, data, config);
  }

  async put<T>(url: string, data?: any, config?: AxiosRequestConfig): Promise<
    AxiosResponse<T>> {
    return this.axiosInstance.put<T>(url, data, config);
  }

  async delete<T>(url: string, config?: AxiosRequestConfig): Promise<AxiosResponse<T
    >> {
    return this.axiosInstance.delete<T>(url, config);
  }
}

// Custom error class
export class HttpError extends Error {
  constructor(
    message: string,
    public readonly status: number,
    public readonly data?: any
  ) {
    super(message);
    this.name = 'HttpError';
  }
}

// Request ID generator
function generateRequestId(): string {
  return `req_${Date.now()}_${Math.random().toString(36).substr(2, 9)}`;
}
```

Listing 7: HTTP Client with Interceptors

5.2 API Response Caching Strategy

```
// hooks/queries/useJupiterQuote.ts - React Query integration
export function useJupiterQuote(
```

```
  params: QuoteParams | null,
  options: UseQueryOptions<JupiterQuote> = {}
) {
  const jupiterService = useService<JupiterService>('jupiterService');

  return useQuery({
    queryKey: ['jupiter-quote', params],
    queryFn: () => {
      if (!params) throw new Error('Quote parameters required');
      return jupiterService.getQuote(params);
    },
    enabled: !!params?.inputMint && !!params?.outputMint && !!params?.amount,
    staleTime: 10000, // 10 seconds
    cacheTime: 60000, // 1 minute
    refetchInterval: 15000, // Refetch every 15 seconds
    refetchOnWindowFocus: true,
    retry: (failureCount, error) => {
      // Don't retry on client errors (4xx)
      if (error instanceof HttpError && error.status >= 400 && error.status < 500) {
        return false;
      }
      return failureCount < 3;
    },
    retryDelay: (attemptIndex) => Math.min(1000 * 2 ** attemptIndex, 30000),
    ...options,
  });
}

// hooks/queries/useTokenBalances.ts - Token balance queries
export function useTokenBalances(publicKey: PublicKey | null) {
  const solanaService = useService<SolanaService>('solanaService');

  return useQueries({
    queries: [
      {
        queryKey: ['token-balance', 'SOL', publicKey?.toString()],
        queryFn: () => solanaService.getSolBalance(publicKey!),
        enabled: !!publicKey,
        staleTime: 30000,
        cacheTime: 300000,
      },
      {
        queryKey: ['token-balance', 'USDC', publicKey?.toString()],
        queryFn: () => solanaService.getTokenBalance(publicKey!, USDC_MINT),
        enabled: !!publicKey,
        staleTime: 30000,
        cacheTime: 300000,
      },
    ],
  });
}

// hooks/mutations/useSwapMutation.ts - Swap execution mutation
export function useSwapMutation() {
  const swapService = useService<SwapService>('swapService');
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: (params: SwapExecutionParams) => swapService.executeSwap(params),
    onSuccess: (result, variables) => {
      // Invalidate relevant queries
      queryClient.invalidateQueries(['token-balance']);
      queryClient.invalidateQueries(['transaction-history']);
```

```
66
67        // Update optimistic data
68        queryClient.setQueryData(
69          ['latest-swap'],
70          {
71            signature: result.signature,
72            timestamp: new Date(),
73            inputToken: variables.inputToken,
74            outputToken: variables.outputToken,
75          }
76        );
77      },
78      onError: (error) => {
79        // Handle error (already logged by service layer)
80        console.error('Swap mutation failed:', error);
81      },
82    });
83  }
```

Listing 8: React Query Integration with Caching

6 Security Implementation

6.1 Input Validation Sanitization

```
1  // utils/validation.ts - Input validation utilities
2  export class ValidationService {
3    static validateTokenAmount(amount: string, decimals: number): ValidationResult {
4      const result: ValidationResult = { isValid: true, errors: [] };
5
6      // Check if empty
7      if (!amount || amount.trim() === '') {
8        result.isValid = false;
9        result.errors.push('Amount is required');
10       return result;
11     }
12
13     // Check if numeric
14     const numericAmount = parseFloat(amount);
15     if (isNaN(numericAmount)) {
16       result.isValid = false;
17       result.errors.push('Amount must be a valid number');
18       return result;
19     }
20
21     // Check if positive
22     if (numericAmount <= 0) {
23       result.isValid = false;
24       result.errors.push('Amount must be greater than zero');
25       return result;
26     }
27
28     // Check decimal places
29     const decimalPlaces = (amount.split('.')[1] || '').length;
30     if (decimalPlaces > decimals) {
31       result.isValid = false;
32       result.errors.push(`Amount cannot have more than ${decimals} decimal places`);
33       return result;
34     }
35
36     // Check reasonable limits
```

```
    const maxAmount = 1000000; // 1M tokens max
    if (numericAmount > maxAmount) {
      result.isValid = false;
      result.errors.push(`Amount cannot exceed ${maxAmount.toLocaleString()}`);
      return result;
    }

    return result;
  }

  static validatePublicKey(publicKey: string): ValidationResult {
    const result: ValidationResult = { isValid: true, errors: [] };

    try {
      new PublicKey(publicKey);
    } catch (error) {
      result.isValid = false;
      result.errors.push('Invalid public key format');
    }

    return result;
  }

  static validateSlippage(slippage: number): ValidationResult {
    const result: ValidationResult = { isValid: true, errors: [] };

    if (slippage < 0.1 || slippage > 50) {
      result.isValid = false;
      result.errors.push('Slippage must be between 0.1% and 50%');
    }

    return result;
  }

  static sanitizeAmount(amount: string): string {
    // Remove any non-numeric characters except decimal point
    return amount.replace(/[^0-9.]/g, '')
                 .replace(/(\..*)\./g, '$1'); // Remove extra decimal points
  }
}

// Custom validation hook
export function useValidation() {
  const validateAndSanitize = useCallback((
    value: string,
    validator: (value: string) => ValidationResult,
    sanitizer?: (value: string) => string
  ) => {
    const sanitized = sanitizer ? sanitizer(value) : value;
    const validation = validator(sanitized);

    return {
      value: sanitized,
      isValid: validation.isValid,
      errors: validation.errors,
    };
  }, []);

  return { validateAndSanitize };
}
```

Listing 9: Comprehensive Input Validation

6.2 Transaction Security Patterns

```typescript
// services/security/TransactionSecurity.ts
export class TransactionSecurity {
  private readonly logger: Logger;
  private readonly rpcManager: RpcManager;

  constructor(logger: Logger, rpcManager: RpcManager) {
    this.logger = logger;
    this.rpcManager = rpcManager;
  }

  /**
   * Validate transaction before signing
   */
  async validateTransaction(
    transaction: VersionedTransaction,
    expectedParams: TransactionValidationParams
  ): Promise<ValidationResult> {
    const result: ValidationResult = { isValid: true, errors: [] };

    try {
      // 1. Validate transaction structure
      if (!transaction.message || !transaction.signatures) {
        result.isValid = false;
        result.errors.push('Invalid transaction structure');
        return result;
      }

      // 2. Validate account permissions
      const accountValidation = await this.validateAccountPermissions(
        transaction,
        expectedParams.userPublicKey
      );
      if (!accountValidation.isValid) {
        result.errors.push(...accountValidation.errors);
        result.isValid = false;
      }

      // 3. Validate instruction data
      const instructionValidation = this.validateInstructions(
        transaction,
        expectedParams
      );
      if (!instructionValidation.isValid) {
        result.errors.push(...instructionValidation.errors);
        result.isValid = false;
      }

      // 4. Simulate transaction
      const simulationResult = await this.simulateTransaction(transaction);
      if (!simulationResult.success) {
        result.isValid = false;
        result.errors.push(`Simulation failed: ${simulationResult.error}`);
      }

      return result;
    } catch (error) {
      this.logger.error('Transaction validation failed:', error);
      return {
        isValid: false,
        errors: ['Transaction validation failed'],
      };
```

```
    }
  }

  /**
   * Simulate transaction before execution
   */
  private async simulateTransaction(
    transaction: VersionedTransaction
  ): Promise<{ success: boolean; error?: string }> {
    try {
      const connection = this.rpcManager.getConnection();
      const simulation = await connection.simulateTransaction(transaction, {
        sigVerify: false,
        replaceRecentBlockhash: true,
      });

      if (simulation.value.err) {
        return {
          success: false,
          error: JSON.stringify(simulation.value.err),
        };
      }

      // Check for suspicious patterns in logs
      const logs = simulation.value.logs || [];
      const suspiciousPatterns = [
        'insufficient funds',
        'custom program error',
        'invalid instruction',
      ];

      for (const pattern of suspiciousPatterns) {
        if (logs.some(log => log.toLowerCase().includes(pattern))) {
          return {
            success: false,
            error: `Suspicious pattern detected: ${pattern}`,
          };
        }
      }

      return { success: true };
    } catch (error) {
      return {
        success: false,
        error: error instanceof Error ? error.message : 'Simulation failed',
      };
    }
  }

  /**
   * Validate account permissions
   */
  private async validateAccountPermissions(
    transaction: VersionedTransaction,
    userPublicKey: PublicKey
  ): Promise<ValidationResult> {
    const result: ValidationResult = { isValid: true, errors: [] };

    // Check if user is a signer
    const accountKeys = transaction.message.staticAccountKeys;
    const userAccountIndex = accountKeys.findIndex(key => key.equals(userPublicKey));

    if (userAccountIndex === -1) {
```

```
      result.isValid = false;
      result.errors.push('User account not found in transaction');
      return result;
    }

    // Validate that user is required signer
    const requiredSigners = transaction.message.header.numRequiredSignatures;
    if (userAccountIndex >= requiredSigners) {
      result.isValid = false;
      result.errors.push('User is not a required signer');
    }

    return result;
  }

  /**
   * Validate transaction instructions
   */
  private validateInstructions(
    transaction: VersionedTransaction,
    expectedParams: TransactionValidationParams
  ): ValidationResult {
    const result: ValidationResult = { isValid: true, errors: [] };

    const instructions = transaction.message.compiledInstructions;

    // Check for expected Jupiter program
    const jupiterProgramId = new PublicKey('
    JUP6LkbZbjS1jKKwapdHNy74zcZ3tLUZoi5QNyVTaV4');
    const hasJupiterInstruction = instructions.some(instruction => {
      const programId = transaction.message.staticAccountKeys[instruction.
    programIdIndex];
      return programId.equals(jupiterProgramId);
    });

    if (!hasJupiterInstruction) {
      result.isValid = false;
      result.errors.push('No Jupiter instruction found');
    }

    // Validate instruction count (reasonable limits)
    if (instructions.length > 10) {
      result.isValid = false;
      result.errors.push('Too many instructions in transaction');
    }

    return result;
  }
}
```

Listing 10: Secure Transaction Handling

7 Performance Optimization

7.1 Code Splitting Strategy

```
// components/lazy/LazyComponents.tsx - Lazy loading setup
import { lazy, Suspense } from 'react';
import { LoadingSpinner } from '@/components/ui/loading-spinner';

// Lazy load heavy components
```

```
export const SwapInterface = lazy(() =>
  import('@/components/swap/SwapInterface').then(module => ({
    default: module.SwapInterface
  }))
);

export const TransactionHistory = lazy(() =>
  import('@/components/analytics/TransactionHistory').then(module => ({
    default: module.TransactionHistory
  }))
);

export const OptimizationPanel = lazy(() =>
  import('@/components/analytics/OptimizationPanel').then(module => ({
    default: module.OptimizationPanel
  }))
);

// Lazy wrapper component
export function LazyWrapper({
  children,
  fallback = <LoadingSpinner />
}: {
  children: React.ReactNode;
  fallback?: React.ReactNode;
}) {
  return (
    <Suspense fallback={fallback}>
      {children}
    </Suspense>
  );
}

// Route-based code splitting
export const routes = [
  {
    path: '/',
    component: lazy(() => import('@/app/page')),
  },
  {
    path: '/analytics',
    component: lazy(() => import('@/app/analytics/page')),
  },
  {
    path: '/history',
    component: lazy(() => import('@/app/history/page')),
  },
];
```

Listing 11: Advanced Code Splitting Implementation

7.2 Performance Monitoring

```
// utils/performance.ts - Performance monitoring utilities
export class PerformanceMonitor {
  private static instance: PerformanceMonitor;
  private metrics: Map<string, PerformanceMetric> = new Map();

  static getInstance(): PerformanceMonitor {
    if (!PerformanceMonitor.instance) {
      PerformanceMonitor.instance = new PerformanceMonitor();
    }
```

```
    return PerformanceMonitor.instance;
  }

  startTiming(name: string): void {
    this.metrics.set(name, {
      name,
      startTime: performance.now(),
      endTime: null,
      duration: null,
    });
  }

  endTiming(name: string): number | null {
    const metric = this.metrics.get(name);
    if (!metric) return null;

    metric.endTime = performance.now();
    metric.duration = metric.endTime - metric.startTime;

    // Log to console in development
    if (process.env.NODE_ENV === 'development') {
      console.log(`       ${name}: ${metric.duration.toFixed(2)}ms`);
    }

    // Send to analytics in production
    if (process.env.NODE_ENV === 'production') {
      this.sendToAnalytics(metric);
    }

    return metric.duration;
  }

  measureAsync<T>(name: string, fn: () => Promise<T>): Promise<T> {
    this.startTiming(name);
    return fn().finally(() => {
      this.endTiming(name);
    });
  }

  measureSync<T>(name: string, fn: () => T): T {
    this.startTiming(name);
    try {
      return fn();
    } finally {
      this.endTiming(name);
    }
  }

  private sendToAnalytics(metric: PerformanceMetric): void {
    // Send to your analytics service
    if (typeof window !== 'undefined' && window.gtag) {
      window.gtag('event', 'timing_complete', {
        name: metric.name,
        value: Math.round(metric.duration!),
      });
    }
  }

  getMetrics(): PerformanceMetric[] {
    return Array.from(this.metrics.values());
  }

  clearMetrics(): void {
```

```
    this.metrics.clear();
  }
}

// React hook for performance monitoring
export function usePerformanceMonitor() {
  const monitor = PerformanceMonitor.getInstance();

  const measureRender = useCallback((componentName: string) => {
    useEffect(() => {
      monitor.startTiming(`render_${componentName}`);
      return () => {
        monitor.endTiming(`render_${componentName}`);
      };
    }, [componentName]);
  }, [monitor]);

  const measureAsync = useCallback(
    <T>(name: string, fn: () => Promise<T>) => monitor.measureAsync(name, fn),
    [monitor]
  );

  const measureSync = useCallback(
    <T>(name: string, fn: () => T) => monitor.measureSync(name, fn),
    [monitor]
  );

  return { measureRender, measureAsync, measureSync };
}

// Usage example
export function SwapInterface() {
  const { measureRender, measureAsync } = usePerformanceMonitor();

  // Measure component render time
  measureRender('SwapInterface');

  const handleSwap = useCallback(async () => {
    await measureAsync('swap_execution', async () => {
      // Swap logic here
    });
  }, [measureAsync]);

  return (
    // Component JSX
  );
}
```

Listing 12: Performance Monitoring Implementation

8 Testing Architecture

8.1 Test Structure Organization

```
// __tests__/setup.ts - Global test setup
import '@testing-library/jest-dom';
import { configure } from '@testing-library/react';
import { server } from './mocks/server';

// Configure Testing Library
configure({ testIdAttribute: 'data-testid' });
```

```
8
9  // Setup MSW
10 beforeAll(() => server.listen());
11 afterEach(() => server.resetHandlers());
12 afterAll(() => server.close());
13
14 // Mock environment variables
15 process.env.NEXT_PUBLIC_JUPITER_API_BASE = 'https://quote-api.jup.ag';
16 process.env.NEXT_PUBLIC_HELIUS_API_KEY = 'test_helius_key';
17 process.env.NEXT_PUBLIC_ALCHEMY_API_KEY = 'test_alchemy_key';
18
19 // Mock Solana Web3
20 jest.mock('@solana/web3.js', () => ({
21   ...jest.requireActual('@solana/web3.js'),
22   Connection: jest.fn().mockImplementation(() => ({
23     getLatestBlockhash: jest.fn().mockResolvedValue({
24       blockhash: 'mock_blockhash',
25       lastValidBlockHeight: 123456789,
26     }),
27     getBalance: jest.fn().mockResolvedValue(1000000000),
28     simulateTransaction: jest.fn().mockResolvedValue({
29       value: { err: null, logs: [] },
30     }),
31     sendTransaction: jest.fn().mockResolvedValue('mock_signature'),
32   })),
33 }));
34
35 // __tests__/utils/test-utils.tsx - Custom render function
36 import { render, RenderOptions } from '@testing-library/react';
37 import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
38 import { WalletAdapterNetwork } from '@solana/wallet-adapter-base';
39 import { ConnectionProvider, WalletProvider } from '@solana/wallet-adapter-react';
40
41 interface CustomRenderOptions extends Omit<RenderOptions, 'wrapper'> {
42   queryClient?: QueryClient;
43   walletNetwork?: WalletAdapterNetwork;
44 }
45
46 export function renderWithProviders(
47   ui: React.ReactElement,
48   options: CustomRenderOptions = {}
49 ) {
50   const {
51     queryClient = new QueryClient({
52       defaultOptions: {
53         queries: { retry: false },
54         mutations: { retry: false },
55       },
56     }),
57     walletNetwork = WalletAdapterNetwork.Devnet,
58     ...renderOptions
59   } = options;
60
61   function Wrapper({ children }: { children: React.ReactNode }) {
62     return (
63       <ConnectionProvider endpoint="https://api.devnet.solana.com">
64         <WalletProvider wallets={[]} autoConnect={false}>
65           <QueryClientProvider client={queryClient}>
66             {children}
67           </QueryClientProvider>
68         </WalletProvider>
69       </ConnectionProvider>
70     );
```

```
  }

  return render(ui, { wrapper: Wrapper, ...renderOptions });
}

// Re-export everything
export * from '@testing-library/react';
export { renderWithProviders as render };
```

Listing 13: Comprehensive Test Architecture

9 Conclusion

This comprehensive architecture guide provides a complete overview of the Jupiter Swap DApp's technical implementation. The architecture follows modern best practices for React/Next.js applications while addressing the specific challenges of DeFi and blockchain integration.

9.1 Key Architectural Strengths

Architecture Highlights:

- **Modular Design:** Clear separation of concerns with well-defined boundaries
- **Type Safety:** Comprehensive TypeScript implementation
- **Performance:** Optimized bundle splitting and lazy loading
- **Security:** Multi-layer security with comprehensive validation
- **Testability:** Comprehensive testing strategy with high coverage
- **Maintainability:** Clean code with extensive documentation
- **Scalability:** Architecture supports future growth and features

Architecture designed and implemented by Kamel (@treizeb__)
DeAura.io - July 2025