

API Integration Guide

Jupiter Swap DApp

Implementation Guide

Comprehensive API Integration Documentation

Jupiter API: v6 Integration
Helius RPC: Advanced Features
Alchemy: Backup & Scaling
Coingecko: Market Data

Sentry: Error Monitoring
Custom APIs: Internal Services
WebSocket: Real-time Updates
GraphQL: Advanced Queries

Integration Highlights

Jupiter API v6 Complete Integration
Multi-RPC Endpoint Management
Advanced Error Handling Patterns
Performance Optimization Strategies
Security Best Practices
Rate Limiting & Caching
Real-time Data Streaming
Comprehensive Testing Coverage

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

Contents

1	Jupiter API v6 Integration	2
1.1	Complete Jupiter Service Implementation	2
2	Helius RPC Integration	8
2.1	Advanced Helius Features	8
3	Alchemy Integration	15
3.1	Alchemy as Backup and Scaling Solution	15
4	CoinGecko Market Data Integration	19
4.1	Market Data Service	19
5	Rate Limiting Caching	25
5.1	Advanced Rate Limiting Implementation	25
6	Conclusion	31
6.1	Integration Summary	31

1 Jupiter API v6 Integration

1.1 Complete Jupiter Service Implementation

```
1 /**
2  * Jupiter API v6 Service - Complete Implementation
3  * Handles all Jupiter API interactions with advanced features
4  */
5 export class JupiterService extends BaseService {
6   private readonly apiBase: string;
7   private readonly version: string;
8   private readonly httpClient: HttpClient;
9   private readonly rateLimiter: RateLimiter;
10  private readonly cache: CacheManager;
11
12  constructor(
13    logger: Logger,
14    errorHandler: ErrorHandler,
15    httpClient: HttpClient,
16    rateLimiter: RateLimiter,
17    cache: CacheManager
18  ) {
19    super(logger, errorHandler);
20    this.apiBase = this.validateRequired(
21      process.env.NEXT_PUBLIC_JUPITER_API_BASE,
22      'NEXT_PUBLIC_JUPITER_API_BASE'
23    );
24    this.version = process.env.NEXT_PUBLIC_JUPITER_API_VERSION || 'v6';
25    this.httpClient = httpClient;
26    this.rateLimiter = rateLimiter;
27    this.cache = cache;
28  }
29
30  /**
31   * Get optimized quote with advanced parameters
32   * @param params - Enhanced quote parameters
33   * @returns Promise<JupiterQuote>
34   */
35  async getQuote(params: EnhancedQuoteParams): Promise<JupiterQuote> {
36    this.logger.info('Fetching Jupiter quote', { params });
37
38    // Rate limiting
39    await this.rateLimiter.waitForToken('jupiter-quote');
40
41    // Cache key generation
42    const cacheKey = this.generateQuoteCacheKey(params);
43
44    // Check cache first
45    const cachedQuote = await this.cache.get<JupiterQuote>(cacheKey);
46    if (cachedQuote && !this.isQuoteStale(cachedQuote)) {
47      this.logger.debug('Returning cached quote', { cacheKey });
48      return cachedQuote;
49    }
50
51    return this.executeWithRetry(async () => {
52      const queryParams = this.buildAdvancedQuoteParams(params);
53
54      const response = await this.httpClient.get(
55        `${this.apiBase}/${this.version}/quote`,
56        {
57          params: queryParams,
58          timeout: 10000,
59          headers: {
```

```

60         'User-Agent': 'Jupiter-Swap-DApp/1.0',
61         'Accept-Encoding': 'gzip, deflate',
62     }
63 }
64 );
65
66     const quote = this.validateAndEnhanceQuote(response.data, params);
67
68     // Cache the result
69     await this.cache.set(cacheKey, quote, 15000); // 15 seconds TTL
70
71     return quote;
72 }, 3, 1000);
73 }
74
75 /**
76  * Get swap transaction with advanced options
77  * @param quote - Jupiter quote
78  * @param userPublicKey - User's wallet public key
79  * @param options - Advanced swap options
80  * @returns Promise<SwapTransaction>
81  */
82 async getSwapTransaction(
83     quote: JupiterQuote,
84     userPublicKey: PublicKey,
85     options: AdvancedSwapOptions = {}
86 ): Promise<SwapTransaction> {
87     this.logger.info('Getting swap transaction', {
88         quote: quote.inputMint + '->' + quote.outputMint,
89         userPublicKey: userPublicKey.toString(),
90         options
91     });
92
93     // Rate limiting
94     await this.rateLimiter.waitForToken('jupiter-swap');
95
96     return this.executeWithRetry(async () => {
97         const swapRequest = this.buildAdvancedSwapRequest(quote, userPublicKey, options
98     );
99
100     const response = await this.httpClient.post(
101         `${this.apiBase}/${this.version}/swap`,
102         swapRequest,
103         {
104             timeout: 15000,
105             headers: {
106                 'Content-Type': 'application/json',
107                 'User-Agent': 'Jupiter-Swap-DApp/1.0',
108             }
109         }
110     );
111
112     return this.validateSwapTransaction(response.data);
113 }, 2, 2000);
114 }
115
116 /**
117  * Get supported tokens with metadata
118  * @returns Promise<TokenInfo[]>
119  */
120 async getSupportedTokens(): Promise<TokenInfo[]> {
121     const cacheKey = 'jupiter-tokens';

```

```

122 // Check cache (longer TTL for token list)
123 const cachedTokens = await this.cache.get<TokenInfo[]>(cacheKey);
124 if (cachedTokens) {
125     return cachedTokens;
126 }
127
128 return this.executeWithRetry(async () => {
129     const response = await this.httpClient.get(
130         `${this.apiBase}/${this.version}/tokens`,
131         { timeout: 30000 }
132     );
133
134     const tokens = this.validateTokenList(response.data);
135
136     // Cache for 1 hour
137     await this.cache.set(cacheKey, tokens, 3600000);
138
139     return tokens;
140 });
141 }
142
143 /**
144  * Get route map for advanced routing
145  * @returns Promise<RouteMap>
146  */
147 async getRouteMap(): Promise<RouteMap> {
148     const cacheKey = 'jupiter-route-map';
149
150     const cachedRouteMap = await this.cache.get<RouteMap>(cacheKey);
151     if (cachedRouteMap) {
152         return cachedRouteMap;
153     }
154
155     return this.executeWithRetry(async () => {
156         const response = await this.httpClient.get(
157             `${this.apiBase}/${this.version}/route-map`,
158             { timeout: 30000 }
159         );
160
161         const routeMap = this.validateRouteMap(response.data);
162
163         // Cache for 30 minutes
164         await this.cache.set(cacheKey, routeMap, 1800000);
165
166         return routeMap;
167     });
168 }
169
170 /**
171  * Get price impact analysis
172  * @param params - Price impact parameters
173  * @returns Promise<PriceImpactAnalysis>
174  */
175 async getPriceImpactAnalysis(
176     params: PriceImpactParams
177 ): Promise<PriceImpactAnalysis> {
178     // Get multiple quotes with different amounts to analyze price impact
179     const amounts = [
180         params.baseAmount * 0.1,
181         params.baseAmount * 0.5,
182         params.baseAmount,
183         params.baseAmount * 2,
184         params.baseAmount * 5,

```

```

185 ];
186
187 const quotes = await Promise.all(
188   amounts.map(amount =>
189     this.getQuote({
190       ...params,
191       amount: Math.floor(amount),
192     }).catch(error => {
193       this.logger.warn('Failed to get quote for amount ${amount}:', error);
194       return null;
195     })
196   )
197 );
198
199 return this.analyzePriceImpact(quotes.filter(Boolean) as JupiterQuote[], params);
200 }
201
202 /**
203  * Build advanced quote parameters
204  */
205 private buildAdvancedQuoteParams(params: EnhancedQuoteParams): Record<string,
206   string> {
207   const baseParams = {
208     inputMint: params.inputMint,
209     outputMint: params.outputMint,
210     amount: params.amount.toString(),
211     slippageBps: (params.slippageBps || 50).toString(),
212     feeBps: (params.feeBps || 0).toString(),
213     onlyDirectRoutes: (params.onlyDirectRoutes || false).toString(),
214     asLegacyTransaction: 'false',
215     platformFeeBps: '25', // 0.25% platform fee
216     maxAccounts: '64',
217   };
218
219   // Add advanced parameters
220   if (params.excludeDexes && params.excludeDexes.length > 0) {
221     baseParams['excludeDexes'] = params.excludeDexes.join(',');
222   }
223
224   if (params.onlyDirectRoutes !== undefined) {
225     baseParams['onlyDirectRoutes'] = params.onlyDirectRoutes.toString();
226   }
227
228   if (params.maxSplits !== undefined) {
229     baseParams['maxSplits'] = params.maxSplits.toString();
230   }
231
232   if (params.minimizeSlippage) {
233     baseParams['minimizeSlippage'] = 'true';
234   }
235
236   return baseParams;
237 }
238
239 /**
240  * Build advanced swap request
241  */
242 private buildAdvancedSwapRequest(
243   quote: JupiterQuote,
244   userPublicKey: PublicKey,
245   options: AdvancedSwapOptions
246 ): SwapRequest {
247   const baseRequest = {

```

```

247     quoteResponse: quote,
248     userPublicKey: userPublicKey.toString(),
249     wrapAndUnwrapSol: true,
250     useSharedAccounts: true,
251     feeAccount: options.feeAccount,
252     trackingAccount: options.trackingAccount,
253     computeUnitPriceMicroLamports: options.priorityFee || 'auto',
254     asLegacyTransaction: false,
255     useTokenLedger: false,
256     destinationTokenAccount: options.destinationTokenAccount,
257   };
258
259   // Add advanced options
260   if (options.dynamicComputeUnitLimit) {
261     baseRequest['dynamicComputeUnitLimit'] = true;
262   }
263
264   if (options.skipUserAccountsRpcCalls) {
265     baseRequest['skipUserAccountsRpcCalls'] = true;
266   }
267
268   return baseRequest;
269 }
270
271 /**
272  * Validate and enhance quote response
273  */
274 private validateAndEnhanceQuote(
275   data: any,
276   params: EnhancedQuoteParams
277 ): JupiterQuote {
278   if (!data.inputMint || !data.outputMint || !data.inAmount || !data.outAmount) {
279     throw new ValidationException('Invalid quote response structure');
280   }
281
282   // Enhance quote with additional metadata
283   const enhancedQuote: JupiterQuote = {
284     ...data,
285     timestamp: Date.now(),
286     requestParams: params,
287     priceImpact: this.calculatePriceImpact(data),
288     estimatedGas: this.estimateGasCost(data),
289     routeQuality: this.assessRouteQuality(data),
290   };
291
292   return enhancedQuote;
293 }
294
295 /**
296  * Generate cache key for quote
297  */
298 private generateQuoteCacheKey(params: EnhancedQuoteParams): string {
299   const keyData = {
300     inputMint: params.inputMint,
301     outputMint: params.outputMint,
302     amount: params.amount,
303     slippageBps: params.slippageBps || 50,
304     excludeDexes: params.excludeDexes?.sort() || [],
305   };
306
307   return `jupiter-quote:${Buffer.from(JSON.stringify(keyData)).toString('base64')}`;
308 }

```

```

309
310 /**
311  * Check if quote is stale
312  */
313 private isQuoteStale(quote: JupiterQuote): boolean {
314     const maxAge = 15000; // 15 seconds
315     return Date.now() - quote.timestamp > maxAge;
316 }
317
318 /**
319  * Calculate price impact
320  */
321 private calculatePriceImpact(quote: any): number {
322     // Implementation depends on quote structure
323     // This is a simplified calculation
324     const inputAmount = parseFloat(quote.inAmount);
325     const outputAmount = parseFloat(quote.outAmount);
326     const marketPrice = quote.marketPrice || (outputAmount / inputAmount);
327
328     const executionPrice = outputAmount / inputAmount;
329     const priceImpact = ((marketPrice - executionPrice) / marketPrice) * 100;
330
331     return Math.max(0, priceImpact);
332 }
333
334 /**
335  * Estimate gas cost
336  */
337 private estimateGasCost(quote: any): number {
338     // Base gas cost estimation based on route complexity
339     const baseGas = 5000; // Base transaction cost
340     const routeComplexity = quote.routePlan?.length || 1;
341     const complexityMultiplier = Math.min(routeComplexity * 1000, 10000);
342
343     return baseGas + complexityMultiplier;
344 }
345
346 /**
347  * Assess route quality
348  */
349 private assessRouteQuality(quote: any): 'excellent' | 'good' | 'fair' | 'poor' {
350     const priceImpact = this.calculatePriceImpact(quote);
351     const routeLength = quote.routePlan?.length || 1;
352
353     if (priceImpact < 0.1 && routeLength <= 2) return 'excellent';
354     if (priceImpact < 0.5 && routeLength <= 3) return 'good';
355     if (priceImpact < 1.0 && routeLength <= 4) return 'fair';
356     return 'poor';
357 }
358
359 /**
360  * Analyze price impact across different amounts
361  */
362 private analyzePriceImpact(
363     quotes: JupiterQuote[],
364     params: PriceImpactParams
365 ): PriceImpactAnalysis {
366     const analysis: PriceImpactAnalysis = {
367         baseAmount: params.baseAmount,
368         quotes: quotes.map(quote => ({
369             amount: parseFloat(quote.inAmount),
370             priceImpact: quote.priceImpact || 0,
371             outputAmount: parseFloat(quote.outAmount),

```



```

372     rate: parseFloat(quote.outAmount) / parseFloat(quote.inAmount),
373   })),
374   recommendations: [],
375 };
376
377 // Generate recommendations based on analysis
378 const impacts = analysis.quotes.map(q => q.priceImpact);
379 const avgImpact = impacts.reduce((a, b) => a + b, 0) / impacts.length;
380
381 if (avgImpact > 2.0) {
382   analysis.recommendations.push('Consider splitting large trades into smaller
chunks');
383 }
384
385 if (impacts[impacts.length - 1] > impacts[0] * 3) {
386   analysis.recommendations.push('Price impact increases significantly with larger
amounts');
387 }
388
389 return analysis;
390 }
391 }

```

Listing 1: Advanced Jupiter API v6 Service

2 Helius RPC Integration

2.1 Advanced Helius Features

```

1 /**
2  * Helius RPC Service - Advanced Integration
3  * Leverages Helius-specific features for enhanced performance
4  */
5 export class HeliusService extends BaseService {
6   private readonly apiKey: string;
7   private readonly endpoint: string;
8   private readonly httpClient: HttpClient;
9   private readonly websocketManager: WebSocketManager;
10
11   constructor(
12     logger: Logger,
13     errorHandler: ErrorHandler,
14     httpClient: HttpClient,
15     websocketManager: WebSocketManager
16   ) {
17     super(logger, errorHandler);
18     this.apiKey = this.validateRequired(
19       process.env.NEXT_PUBLIC_HELIUS_API_KEY,
20       'NEXT_PUBLIC_HELIUS_API_KEY'
21     );
22     this.endpoint = 'https://mainnet.helius-rpc.com/?api-key=${this.apiKey}';
23     this.httpClient = httpClient;
24     this.websocketManager = websocketManager;
25   }
26
27   /**
28    * Get enhanced account info with Helius metadata
29    * @param publicKey - Account public key
30    * @returns Promise<EnhancedAccountInfo>
31    */
32   async getEnhancedAccountInfo(publicKey: PublicKey): Promise<EnhancedAccountInfo> {

```

```

33     return this.executeWithRetry(async () => {
34         const response = await this.httpClient.post(this.endpoint, {
35             jsonrpc: '2.0',
36             id: 1,
37             method: 'getAccountInfo',
38             params: [
39                 publicKey.toString(),
40                 {
41                     encoding: 'jsonParsed',
42                     commitment: 'confirmed',
43                 },
44             ],
45         });
46
47         if (response.data.error) {
48             throw new RpcError(response.data.error.message, this.endpoint);
49         }
50
51         // Enhance with Helius-specific data
52         const accountInfo = response.data.result;
53         const enhancedInfo = await this.enhanceAccountInfo(accountInfo, publicKey);
54
55         return enhancedInfo;
56     });
57 }
58
59 /**
60  * Get token accounts with enhanced metadata
61  * @param owner - Owner public key
62  * @returns Promise<EnhancedTokenAccount[]>
63  */
64 async getTokenAccountsByOwner(owner: PublicKey): Promise<EnhancedTokenAccount[]> {
65     return this.executeWithRetry(async () => {
66         const response = await this.httpClient.post(this.endpoint, {
67             jsonrpc: '2.0',
68             id: 1,
69             method: 'getTokenAccountsByOwner',
70             params: [
71                 owner.toString(),
72                 { programId: TOKEN_PROGRAM_ID.toString() },
73                 {
74                     encoding: 'jsonParsed',
75                     commitment: 'confirmed',
76                 },
77             ],
78         });
79
80         if (response.data.error) {
81             throw new RpcError(response.data.error.message, this.endpoint);
82         }
83
84         const accounts = response.data.result.value;
85
86         // Enhance each account with metadata
87         const enhancedAccounts = await Promise.all(
88             accounts.map((account: any) => this.enhanceTokenAccount(account))
89         );
90
91         return enhancedAccounts;
92     });
93 }
94
95 /**

```

```

96  * Get transaction history with enhanced parsing
97  * @param address - Address to get history for
98  * @param options - History options
99  * @returns Promise<EnhancedTransaction[]>
100  */
101  async getTransactionHistory(
102    address: PublicKey,
103    options: TransactionHistoryOptions = {}
104  ): Promise<EnhancedTransaction[]> {
105    return this.executeWithRetry(async () => {
106      const response = await this.httpClient.post(this.endpoint, {
107        jsonrpc: '2.0',
108        id: 1,
109        method: 'getSignaturesForAddress',
110        params: [
111          address.toString(),
112          {
113            limit: options.limit || 50,
114            before: options.before,
115            until: options.until,
116            commitment: 'confirmed',
117          },
118        ],
119      });
120
121      if (response.data.error) {
122        throw new RpcError(response.data.error.message, this.endpoint);
123      }
124
125      const signatures = response.data.result;
126
127      // Get detailed transaction info for each signature
128      const transactions = await this.getTransactionDetails(signatures);
129
130      // Parse and enhance transactions
131      const enhancedTransactions = await Promise.all(
132        transactions.map((tx: any) => this.enhanceTransaction(tx))
133      );
134
135      return enhancedTransactions;
136    });
137  }
138
139  /**
140   * Subscribe to account changes via WebSocket
141   * @param publicKey - Account to monitor
142   * @param callback - Callback for updates
143   * @returns Subscription ID
144   */
145  async subscribeToAccount(
146    publicKey: PublicKey,
147    callback: (accountInfo: EnhancedAccountInfo) => void
148  ): Promise<string> {
149    const wsEndpoint = this.endpoint.replace('https://', 'wss://').replace('http://', 'ws://');
150
151    const subscriptionId = await this.websocketManager.subscribe(
152      wsEndpoint,
153      'accountSubscribe',
154      [
155        publicKey.toString(),
156        {
157          encoding: 'jsonParsed',

```

```

158         commitment: 'confirmed',
159     },
160 ],
161     async (data: any) => {
162         const enhancedInfo = await this.enhanceAccountInfo(data.result, publicKey);
163         callback(enhancedInfo);
164     }
165 );
166
167 this.logger.info('Subscribed to account updates', {
168     publicKey: publicKey.toString(),
169     subscriptionId
170 });
171
172 return subscriptionId;
173 }
174
175 /**
176  * Get DAS (Digital Asset Standard) API data
177  * @param assetId - Asset ID
178  * @returns Promise<DASAsset>
179  */
180 async getDASAsset(assetId: string): Promise<DASAsset> {
181     const dasEndpoint = 'https://mainnet.helius-rpc.com/?api-key=${this.apiKey}';
182
183     return this.executeWithRetry(async () => {
184         const response = await this.httpClient.post(dasEndpoint, {
185             jsonrpc: '2.0',
186             id: 1,
187             method: 'getAsset',
188             params: {
189                 id: assetId,
190             },
191         });
192
193         if (response.data.error) {
194             throw new RpcError(response.data.error.message, dasEndpoint);
195         }
196
197         return response.data.result;
198     });
199 }
200
201 /**
202  * Get assets by owner using DAS API
203  * @param owner - Owner public key
204  * @param options - Query options
205  * @returns Promise<DASAsset[]>
206  */
207 async getAssetsByOwner(
208     owner: PublicKey,
209     options: AssetQueryOptions = {}
210 ): Promise<DASAsset[]> {
211     const dasEndpoint = 'https://mainnet.helius-rpc.com/?api-key=${this.apiKey}';
212
213     return this.executeWithRetry(async () => {
214         const response = await this.httpClient.post(dasEndpoint, {
215             jsonrpc: '2.0',
216             id: 1,
217             method: 'getAssetsByOwner',
218             params: {
219                 ownerAddress: owner.toString(),
220                 page: options.page || 1,

```

```

221         limit: options.limit || 1000,
222         displayOptions: {
223             showFungible: options.showFungible || false,
224             showNativeBalance: options.showNativeBalance || true,
225         },
226     },
227 });
228
229 if (response.data.error) {
230     throw new RpcError(response.data.error.message, dasEndpoint);
231 }
232
233 return response.data.result.items;
234 });
235 }
236
237 /**
238  * Enhance account info with metadata
239  */
240 private async enhanceAccountInfo(
241     accountInfo: any,
242     publicKey: PublicKey
243 ): Promise<EnhancedAccountInfo> {
244     const enhanced: EnhancedAccountInfo = {
245         ...accountInfo,
246         publicKey: publicKey.toString(),
247         enhancedAt: Date.now(),
248     };
249
250     // Add token metadata if it's a token account
251     if (accountInfo?.value?.data?.parsed?.type === 'account') {
252         const tokenMint = accountInfo.value.data.parsed.info.mint;
253         enhanced.tokenMetadata = await this.getTokenMetadata(tokenMint);
254     }
255
256     return enhanced;
257 }
258
259 /**
260  * Enhance token account with metadata
261  */
262 private async enhanceTokenAccount(account: any): Promise<EnhancedTokenAccount> {
263     const enhanced: EnhancedTokenAccount = {
264         ...account,
265         enhancedAt: Date.now(),
266     };
267
268     const tokenMint = account.account.data.parsed.info.mint;
269     enhanced.tokenMetadata = await this.getTokenMetadata(tokenMint);
270
271     return enhanced;
272 }
273
274 /**
275  * Get token metadata
276  */
277 private async getTokenMetadata(mint: string): Promise<TokenMetadata | null> {
278     try {
279         const asset = await this.getDASAsset(mint);
280
281         return {
282             name: asset.content?.metadata?.name || 'Unknown Token',
283             symbol: asset.content?.metadata?.symbol || 'UNKNOWN',

```

```

284         decimals: asset.token_info?.decimals || 0,
285         logoURI: asset.content?.files?.[0]?.uri,
286         description: asset.content?.metadata?.description,
287         tags: asset.grouping?.map(g => g.group_value) || [],
288     };
289     } catch (error) {
290         this.logger.warn('Failed to get metadata for token ${mint}:', error);
291         return null;
292     }
293 }
294
295 /**
296  * Get detailed transaction information
297  */
298 private async getTransactionDetails(signatures: any[]): Promise<any[]> {
299     const batchSize = 10;
300     const batches = [];
301
302     for (let i = 0; i < signatures.length; i += batchSize) {
303         const batch = signatures.slice(i, i + batchSize);
304         batches.push(batch);
305     }
306
307     const allTransactions = [];
308
309     for (const batch of batches) {
310         const batchPromises = batch.map(sig =>
311             this.httpClient.post(this.endpoint, {
312                 jsonrpc: '2.0',
313                 id: 1,
314                 method: 'getTransaction',
315                 params: [
316                     sig.signature,
317                     {
318                         encoding: 'jsonParsed',
319                         commitment: 'confirmed',
320                         maxSupportedTransactionVersion: 0,
321                     },
322                 ],
323             })
324         );
325
326         const batchResults = await Promise.allSettled(batchPromises);
327         const validResults = batchResults
328             .filter(result => result.status === 'fulfilled')
329             .map(result => (result as PromiseFulfilledResult<any>).value.data.result)
330             .filter(result => result && !result.error);
331
332         allTransactions.push(...validResults);
333     }
334
335     return allTransactions;
336 }
337
338 /**
339  * Enhance transaction with parsed data
340  */
341 private async enhanceTransaction(transaction: any): Promise<EnhancedTransaction> {
342     const enhanced: EnhancedTransaction = {
343         signature: transaction.transaction.signatures[0],
344         slot: transaction.slot,
345         blockTime: transaction.blockTime,
346         confirmationStatus: transaction.confirmationStatus,

```

```

347     err: transaction.meta?.err,
348     fee: transaction.meta?.fee,
349     enhancedAt: Date.now(),
350     parsedInstructions: [],
351     tokenTransfers: [],
352     solTransfers: [],
353   };
354
355   // Parse instructions
356   if (transaction.transaction?.message?.instructions) {
357     enhanced.parsedInstructions = await this.parseInstructions(
358       transaction.transaction.message.instructions
359     );
360   }
361
362   // Extract token transfers
363   if (transaction.meta?.postTokenBalances && transaction.meta?.preTokenBalances) {
364     enhanced.tokenTransfers = this.extractTokenTransfers(
365       transaction.meta.preTokenBalances,
366       transaction.meta.postTokenBalances
367     );
368   }
369
370   // Extract SOL transfers
371   if (transaction.meta?.postBalances && transaction.meta?.preBalances) {
372     enhanced.solTransfers = this.extractSolTransfers(
373       transaction.meta.preBalances,
374       transaction.meta.postBalances,
375       transaction.transaction.message.accountKeys
376     );
377   }
378
379   return enhanced;
380 }
381
382 /**
383  * Parse transaction instructions
384  */
385 private async parseInstructions(instructions: any[]): Promise<ParsedInstruction[]>
386 {
387   return instructions.map(instruction => ({
388     programId: instruction.programId,
389     parsed: instruction.parsed || null,
390     program: instruction.program || 'unknown',
391     type: instruction.parsed?.type || 'unknown',
392     info: instruction.parsed?.info || {},
393   }));
394 }
395
396 /**
397  * Extract token transfers from balance changes
398  */
399 private extractTokenTransfers(
400   preBalances: any[],
401   postBalances: any[]
402 ): TokenTransfer[] {
403   const transfers: TokenTransfer[] = [];
404
405   // Create maps for easier lookup
406   const preMap = new Map(preBalances.map(b => [b.accountIndex, b]));
407   const postMap = new Map(postBalances.map(b => [b.accountIndex, b]));
408
409   // Find all account indices that had balance changes

```

```

409     const allIndices = new Set([...preMap.keys(), ...postMap.keys()]);
410
411     for (const index of allIndices) {
412         const pre = preMap.get(index);
413         const post = postMap.get(index);
414
415         if (pre && post && pre.mint === post.mint) {
416             const preAmount = parseFloat(pre.uiTokenAmount.amount);
417             const postAmount = parseFloat(post.uiTokenAmount.amount);
418             const change = postAmount - preAmount;
419
420             if (change !== 0) {
421                 transfers.push({
422                     mint: pre.mint,
423                     amount: Math.abs(change),
424                     decimals: pre.uiTokenAmount.decimals,
425                     direction: change > 0 ? 'in' : 'out',
426                     accountIndex: index,
427                 });
428             }
429         }
430     }
431
432     return transfers;
433 }
434
435 /**
436  * Extract SOL transfers from balance changes
437  */
438 private extractSolTransfers(
439     preBalances: number[],
440     postBalances: number[],
441     accountKeys: string[]
442 ): SolTransfer[] {
443     const transfers: SolTransfer[] = [];
444
445     for (let i = 0; i < preBalances.length; i++) {
446         const change = postBalances[i] - preBalances[i];
447         if (change !== 0) {
448             transfers.push({
449                 account: accountKeys[i],
450                 amount: Math.abs(change),
451                 direction: change > 0 ? 'in' : 'out',
452             });
453         }
454     }
455
456     return transfers;
457 }
458 }

```

Listing 2: Helius RPC Service with Advanced Features

3 Alchemy Integration

3.1 Alchemy as Backup and Scaling Solution

```

1 /**
2  * Alchemy Service - Backup RPC and Enhanced Features
3  * Provides backup RPC functionality and Alchemy-specific features
4  */

```



```

5 export class AlchemyService extends BaseService {
6   private readonly apiKey: string;
7   private readonly endpoint: string;
8   private readonly httpClient: HttpClient;
9
10  constructor(
11    logger: Logger,
12    errorHandler: ErrorHandler,
13    httpClient: HttpClient
14  ) {
15    super(logger, errorHandler);
16    this.apiKey = this.validateRequired(
17      process.env.NEXT_PUBLIC_ALCHEMY_API_KEY,
18      'NEXT_PUBLIC_ALCHEMY_API_KEY'
19    );
20    this.endpoint = 'https://solana-mainnet.g.alchemy.com/v2/${this.apiKey}';
21    this.httpClient = httpClient;
22  }
23
24  /**
25   * Get enhanced token balances using Alchemy's optimized endpoints
26   * @param owner - Owner public key
27   * @returns Promise<AlchemyTokenBalance[]>
28   */
29  async getTokenBalances(owner: PublicKey): Promise<AlchemyTokenBalance[]> {
30    return this.executeWithRetry(async () => {
31      const response = await this.httpClient.post(this.endpoint, {
32        jsonrpc: '2.0',
33        id: 1,
34        method: 'alchemy_getTokenBalances',
35        params: [
36          owner.toString(),
37          {
38            mint: 'all',
39          },
40        ],
41      });
42
43      if (response.data.error) {
44        throw new RpcError(response.data.error.message, this.endpoint);
45      }
46
47      return response.data.result.tokenBalances.map((balance: any) => ({
48        mint: balance.mint,
49        amount: balance.amount,
50        decimals: balance.decimals,
51        uiAmount: balance.uiAmount,
52        uiAmountString: balance.uiAmountString,
53        enhancedAt: Date.now(),
54      }));
55    });
56  }
57
58  /**
59   * Get token metadata using Alchemy's metadata service
60   * @param mints - Array of token mints
61   * @returns Promise<AlchemyTokenMetadata[]>
62   */
63  async getTokenMetadata(mints: string[]): Promise<AlchemyTokenMetadata[]> {
64    return this.executeWithRetry(async () => {
65      const response = await this.httpClient.post(this.endpoint, {
66        jsonrpc: '2.0',
67        id: 1,

```

```

68     method: 'alchemy_getTokenMetadata',
69     params: [
70         {
71             mints: mints,
72         },
73     ],
74 });
75
76 if (response.data.error) {
77     throw new RpcError(response.data.error.message, this.endpoint);
78 }
79
80 return response.data.result.map((metadata: any) => ({
81     mint: metadata.mint,
82     name: metadata.name,
83     symbol: metadata.symbol,
84     decimals: metadata.decimals,
85     logoURI: metadata.logoURI,
86     tags: metadata.tags || [],
87     description: metadata.description,
88     enhancedAt: Date.now(),
89 }));
90 });
91 }
92
93 /**
94  * Get NFTs owned by address
95  * @param owner - Owner public key
96  * @param options - Query options
97  * @returns Promise<AlchemyNFT[]>
98  */
99 async getNFTsByOwner(
100     owner: PublicKey,
101     options: NFTQueryOptions = {}
102 ): Promise<AlchemyNFT[]> {
103     return this.executeWithRetry(async () => {
104         const response = await this.httpClient.post(this.endpoint, {
105             jsonrpc: '2.0',
106             id: 1,
107             method: 'alchemy_getNFTs',
108             params: [
109                 owner.toString(),
110                 {
111                     pageKey: options.pageKey,
112                     limit: options.limit || 100,
113                     withMetadata: options.withMetadata !== false,
114                 },
115             ],
116         });
117
118         if (response.data.error) {
119             throw new RpcError(response.data.error.message, this.endpoint);
120         }
121
122         return response.data.result.nfts.map((nft: any) => ({
123             mint: nft.mint,
124             name: nft.name,
125             symbol: nft.symbol,
126             uri: nft.uri,
127             metadata: nft.metadata,
128             collection: nft.collection,
129             enhancedAt: Date.now(),
130         }));

```

```

131     });
132   }
133
134   /**
135    * Get transaction receipts with enhanced parsing
136    * @param signatures - Transaction signatures
137    * @returns Promise<AlchemyTransactionReceipt[]>
138    */
139   async getTransactionReceipts(
140     signatures: string[]
141   ): Promise<AlchemyTransactionReceipt[]> {
142     const batchSize = 25; // Alchemy's batch limit
143     const batches = [];
144
145     for (let i = 0; i < signatures.length; i += batchSize) {
146       const batch = signatures.slice(i, i + batchSize);
147       batches.push(batch);
148     }
149
150     const allReceipts = [];
151
152     for (const batch of batches) {
153       const batchRequests = batch.map((signature, index) => ({
154         jsonrpc: '2.0',
155         id: index + 1,
156         method: 'getTransaction',
157         params: [
158           signature,
159           {
160             encoding: 'jsonParsed',
161             commitment: 'confirmed',
162             maxSupportedTransactionVersion: 0,
163           },
164         ],
165       }));
166
167       const response = await this.httpClient.post(this.endpoint, batchRequests);
168
169       if (Array.isArray(response.data)) {
170         const validReceipts = response.data
171           .filter(result => result.result && !result.error)
172           .map(result => this.enhanceTransactionReceipt(result.result));
173
174         allReceipts.push(...validReceipts);
175       }
176     }
177
178     return allReceipts;
179   }
180
181   /**
182    * Get account activity with Alchemy's enhanced parsing
183    * @param address - Address to get activity for
184    * @param options - Activity options
185    * @returns Promise<AlchemyActivity[]>
186    */
187   async getAccountActivity(
188     address: PublicKey,
189     options: ActivityOptions = {}
190   ): Promise<AlchemyActivity[]> {
191     return this.executeWithRetry(async () => {
192       const response = await this.httpClient.post(this.endpoint, {
193         jsonrpc: '2.0',

```

```

194         id: 1,
195         method: 'alchemy_getAccountActivity',
196         params: [
197             address.toString(),
198             {
199                 limit: options.limit || 50,
200                 before: options.before,
201                 after: options.after,
202                 activityTypes: options.activityTypes || ['TRANSFER', 'SWAP', 'BURN', '
MINT'],
203             },
204         ],
205     });
206
207     if (response.data.error) {
208         throw new RpcError(response.data.error.message, this.endpoint);
209     }
210
211     return response.data.result.activities.map((activity: any) => ({
212         signature: activity.signature,
213         type: activity.type,
214         source: activity.source,
215         timestamp: activity.timestamp,
216         nativeTransfers: activity.nativeTransfers || [],
217         tokenTransfers: activity.tokenTransfers || [],
218         description: activity.description,
219         enhancedAt: Date.now(),
220     }));
221 });
222 }
223
224 /**
225  * Enhance transaction receipt with additional parsing
226  */
227 private enhanceTransactionReceipt(transaction: any): AlchemyTransactionReceipt {
228     return {
229         signature: transaction.transaction.signatures[0],
230         slot: transaction.slot,
231         blockTime: transaction.blockTime,
232         confirmationStatus: transaction.confirmationStatus,
233         err: transaction.meta?.err,
234         fee: transaction.meta?.fee,
235         computeUnitsConsumed: transaction.meta?.computeUnitsConsumed,
236         logMessages: transaction.meta?.logMessages || [],
237         preBalances: transaction.meta?.preBalances || [],
238         postBalances: transaction.meta?.postBalances || [],
239         preTokenBalances: transaction.meta?.preTokenBalances || [],
240         postTokenBalances: transaction.meta?.postTokenBalances || [],
241         innerInstructions: transaction.meta?.innerInstructions || [],
242         enhancedAt: Date.now(),
243     };
244 }
245 }

```

Listing 3: Alchemy Service Implementation

4 CoinGecko Market Data Integration

4.1 Market Data Service

```

1 /**

```

```

2  * CoinGecko Service - Market Data Integration
3  * Provides comprehensive market data for optimization algorithms
4  */
5  export class CoingeckoService extends BaseService {
6      private readonly apiBase: string;
7      private readonly apiKey?: string;
8      private readonly httpClient: HttpClient;
9      private readonly cache: CacheManager;
10     private readonly rateLimiter: RateLimiter;
11
12     constructor(
13         logger: Logger,
14         errorHandler: ErrorHandler,
15         httpClient: HttpClient,
16         cache: CacheManager,
17         rateLimiter: RateLimiter
18     ) {
19         super(logger, errorHandler);
20         this.apiBase = 'https://api.coingecko.com/api/v3';
21         this.apiKey = process.env.NEXT_PUBLIC_COINGECKO_API_KEY;
22         this.httpClient = httpClient;
23         this.cache = cache;
24         this.rateLimiter = rateLimiter;
25     }
26
27     /**
28      * Get token market data
29      * @param tokenId - CoinGecko token ID
30      * @returns Promise<TokenMarketData>
31      */
32     async getTokenMarketData(tokenId: string): Promise<TokenMarketData> {
33         const cacheKey = `coingecko-market-${tokenId}`;
34
35         // Check cache first (5 minutes TTL)
36         const cached = await this.cache.get<TokenMarketData>(cacheKey);
37         if (cached) {
38             return cached;
39         }
40
41         // Rate limiting
42         await this.rateLimiter.waitForToken('coingecko-api');
43
44         return this.executeWithRetry(async () => {
45             const headers: Record<string, string> = {};
46             if (this.apiKey) {
47                 headers['X-CG-Pro-API-Key'] = this.apiKey;
48             }
49
50             const response = await this.httpClient.get(
51                 `${this.apiBase}/coins/${tokenId}`,
52                 {
53                     params: {
54                         localization: false,
55                         tickers: false,
56                         market_data: true,
57                         community_data: false,
58                         developer_data: false,
59                         sparkline: true,
60                     },
61                     headers,
62                     timeout: 10000,
63                 }
64             );

```

```

65
66     const marketData = this.parseMarketData(response.data);
67
68     // Cache for 5 minutes
69     await this.cache.set(cacheKey, marketData, 300000);
70
71     return marketData;
72   });
73 }
74
75 /**
76  * Get multiple tokens market data
77  * @param tokenIds - Array of CoinGecko token IDs
78  * @returns Promise<TokenMarketData[]>
79  */
80 async getMultipleTokensMarketData(tokenIds: string[]): Promise<TokenMarketData[]> {
81   // Rate limiting
82   await this.rateLimiter.waitForToken('coingecko-api');
83
84   return this.executeWithRetry(async () => {
85     const headers: Record<string, string> = {};
86     if (this.apiKey) {
87       headers['X-CG-Pro-API-Key'] = this.apiKey;
88     }
89
90     const response = await this.httpClient.get(
91       `${this.apiBase}/coins/markets`,
92       {
93         params: {
94           vs_currency: 'usd',
95           ids: tokenIds.join(','),
96           order: 'market_cap_desc',
97           per_page: 250,
98           page: 1,
99           sparkline: true,
100           price_change_percentage: '1h,24h,7d,30d',
101         },
102         headers,
103         timeout: 15000,
104       }
105     );
106
107     return response.data.map((coin: any) => this.parseMarketDataFromList(coin));
108   });
109 }
110
111 /**
112  * Get token price history
113  * @param tokenId - CoinGecko token ID
114  * @param days - Number of days of history
115  * @returns Promise<PriceHistory>
116  */
117 async getTokenPriceHistory(tokenId: string, days: number = 7): Promise<PriceHistory> {
118   > {
119     const cacheKey = `coingecko-history-${tokenId}-${days}`;
120
121     // Check cache (longer TTL for historical data)
122     const cached = await this.cache.get<PriceHistory>(cacheKey);
123     if (cached) {
124       return cached;
125     }
126
127     // Rate limiting

```

```

127     await this.rateLimiter.waitForToken('coingecko-api');
128
129     return this.executeWithRetry(async () => {
130         const headers: Record<string, string> = {};
131         if (this.apiKey) {
132             headers['X-CG-Pro-API-Key'] = this.apiKey;
133         }
134
135         const response = await this.httpClient.get(
136             `${this.apiBase}/coins/${tokenId}/market_chart`,
137             {
138                 params: {
139                     vs_currency: 'usd',
140                     days: days.toString(),
141                     interval: days <= 1 ? 'hourly' : 'daily',
142                 },
143                 headers,
144                 timeout: 15000,
145             }
146         );
147
148         const history = this.parsePriceHistory(response.data);
149
150         // Cache for 1 hour
151         await this.cache.set(cacheKey, history, 3600000);
152
153         return history;
154     });
155 }
156
157 /**
158  * Search for tokens
159  * @param query - Search query
160  * @returns Promise<TokenSearchResult[]>
161  */
162 async searchTokens(query: string): Promise<TokenSearchResult[]> {
163     if (query.length < 2) {
164         return [];
165     }
166
167     const cacheKey = `coingecko-search-${query.toLowerCase()}`;
168
169     // Check cache (10 minutes TTL)
170     const cached = await this.cache.get<TokenSearchResult[]>(cacheKey);
171     if (cached) {
172         return cached;
173     }
174
175     // Rate limiting
176     await this.rateLimiter.waitForToken('coingecko-api');
177
178     return this.executeWithRetry(async () => {
179         const headers: Record<string, string> = {};
180         if (this.apiKey) {
181             headers['X-CG-Pro-API-Key'] = this.apiKey;
182         }
183
184         const response = await this.httpClient.get(
185             `${this.apiBase}/search`,
186             {
187                 params: { query },
188                 headers,
189                 timeout: 10000,

```

```

190     }
191   );
192
193   const results = response.data.coins.slice(0, 20).map((coin: any) => ({
194     id: coin.id,
195     name: coin.name,
196     symbol: coin.symbol,
197     thumb: coin.thumb,
198     large: coin.large,
199     market_cap_rank: coin.market_cap_rank,
200   }));
201
202   // Cache for 10 minutes
203   await this.cache.set(cacheKey, results, 600000);
204
205   return results;
206 });
207 }
208
209 /**
210  * Get global market data
211  * @returns Promise<GlobalMarketData>
212  */
213 async getGlobalMarketData(): Promise<GlobalMarketData> {
214   const cacheKey = 'coingecko-global';
215
216   // Check cache (10 minutes TTL)
217   const cached = await this.cache.get<GlobalMarketData>(cacheKey);
218   if (cached) {
219     return cached;
220   }
221
222   // Rate limiting
223   await this.rateLimiter.waitForToken('coingecko-api');
224
225   return this.executeWithRetry(async () => {
226     const headers: Record<string, string> = {};
227     if (this.apiKey) {
228       headers['X-CG-Pro-API-Key'] = this.apiKey;
229     }
230
231     const response = await this.httpClient.get(
232       `${this.apiBase}/global`,
233       { headers, timeout: 10000 }
234     );
235
236     const globalData = this.parseGlobalData(response.data.data);
237
238     // Cache for 10 minutes
239     await this.cache.set(cacheKey, globalData, 600000);
240
241     return globalData;
242   });
243 }
244
245 /**
246  * Parse market data from detailed coin response
247  */
248 private parseMarketData(data: any): TokenMarketData {
249   const marketData = data.market_data;
250
251   return {
252     id: data.id,

```



```

253     symbol: data.symbol,
254     name: data.name,
255     image: data.image?.large,
256     current_price: marketData.current_price?.usd || 0,
257     market_cap: marketData.market_cap?.usd || 0,
258     market_cap_rank: data.market_cap_rank,
259     fully_diluted_valuation: marketData.fully_diluted_valuation?.usd,
260     total_volume: marketData.total_volume?.usd || 0,
261     high_24h: marketData.high_24h?.usd || 0,
262     low_24h: marketData.low_24h?.usd || 0,
263     price_change_24h: marketData.price_change_24h || 0,
264     price_change_percentage_24h: marketData.price_change_percentage_24h || 0,
265     price_change_percentage_7d: marketData.price_change_percentage_7d || 0,
266     price_change_percentage_30d: marketData.price_change_percentage_30d || 0,
267     circulating_supply: marketData.circulating_supply || 0,
268     total_supply: marketData.total_supply || 0,
269     max_supply: marketData.max_supply,
270     ath: marketData.ath?.usd || 0,
271     ath_change_percentage: marketData.ath_change_percentage?.usd || 0,
272     ath_date: marketData.ath_date?.usd,
273     atl: marketData.atl?.usd || 0,
274     atl_change_percentage: marketData.atl_change_percentage?.usd || 0,
275     atl_date: marketData.atl_date?.usd,
276     last_updated: data.last_updated,
277     sparkline_in_7d: marketData.sparkline_7d?.price || [],
278   };
279 }
280
281 /**
282  * Parse market data from markets list response
283  */
284 private parseMarketDataFromList(data: any): TokenMarketData {
285   return {
286     id: data.id,
287     symbol: data.symbol,
288     name: data.name,
289     image: data.image,
290     current_price: data.current_price || 0,
291     market_cap: data.market_cap || 0,
292     market_cap_rank: data.market_cap_rank,
293     fully_diluted_valuation: data.fully_diluted_valuation,
294     total_volume: data.total_volume || 0,
295     high_24h: data.high_24h || 0,
296     low_24h: data.low_24h || 0,
297     price_change_24h: data.price_change_24h || 0,
298     price_change_percentage_24h: data.price_change_percentage_24h || 0,
299     price_change_percentage_7d: data.price_change_percentage_7d || 0,
300     price_change_percentage_30d: data.price_change_percentage_30d || 0,
301     circulating_supply: data.circulating_supply || 0,
302     total_supply: data.total_supply || 0,
303     max_supply: data.max_supply,
304     ath: data.ath || 0,
305     ath_change_percentage: data.ath_change_percentage || 0,
306     ath_date: data.ath_date,
307     atl: data.atl || 0,
308     atl_change_percentage: data.atl_change_percentage || 0,
309     atl_date: data.atl_date,
310     last_updated: data.last_updated,
311     sparkline_in_7d: data.sparkline_in_7d?.price || [],
312   };
313 }
314
315 /**

```

```

316  * Parse price history data
317  */
318  private parsePriceHistory(data: any): PriceHistory {
319    return {
320      prices: data.prices.map(([timestamp, price]: [number, number]) => ({
321        timestamp,
322        price,
323        date: new Date(timestamp),
324      })),
325      market_caps: data.market_caps.map(([timestamp, cap]: [number, number]) => ({
326        timestamp,
327        market_cap: cap,
328        date: new Date(timestamp),
329      })),
330      total_volumes: data.total_volumes.map(([timestamp, volume]: [number, number])
331      => ({
332        timestamp,
333        volume,
334        date: new Date(timestamp),
335      })),
336    };
337  }
338  /**
339  * Parse global market data
340  */
341  private parseGlobalData(data: any): GlobalMarketData {
342    return {
343      active_cryptocurrencies: data.active_cryptocurrencies,
344      upcoming_icos: data.upcoming_icos,
345      ongoing_icos: data.ongoing_icos,
346      ended_icos: data.ended_icos,
347      markets: data.markets,
348      total_market_cap: data.total_market_cap?.usd || 0,
349      total_volume: data.total_volume?.usd || 0,
350      market_cap_percentage: data.market_cap_percentage,
351      market_cap_change_percentage_24h_usd: data.market_cap_change_percentage_24h_usd
352      || 0,
353      updated_at: data.updated_at,
354    };
355  }

```

Listing 4: CoinGecko Service for Market Data

5 Rate Limiting Caching

5.1 Advanced Rate Limiting Implementation

```

1  /**
2  * Rate Limiter - Token Bucket Algorithm Implementation
3  * Provides sophisticated rate limiting for API calls
4  */
5  export class RateLimiter {
6    private buckets = new Map<string, TokenBucket>();
7    private readonly logger: Logger;
8
9    constructor(logger: Logger) {
10      this.logger = logger;
11    }
12

```

```

13  /**
14   * Configure rate limit for a specific endpoint
15   * @param key - Unique identifier for the endpoint
16   * @param config - Rate limit configuration
17   */
18  configure(key: string, config: RateLimitConfig): void {
19    this.buckets.set(key, new TokenBucket(config, this.logger));
20    this.logger.info('Rate limiter configured', { key, config });
21  }
22
23  /**
24   * Wait for available token
25   * @param key - Endpoint identifier
26   * @returns Promise<void>
27   */
28  async waitForToken(key: string): Promise<void> {
29    const bucket = this.buckets.get(key);
30    if (!bucket) {
31      // No rate limit configured, allow immediately
32      return;
33    }
34
35    await bucket.consume();
36  }
37
38  /**
39   * Check if request would be allowed without consuming token
40   * @param key - Endpoint identifier
41   * @returns boolean
42   */
43  canMakeRequest(key: string): boolean {
44    const bucket = this.buckets.get(key);
45    if (!bucket) {
46      return true;
47    }
48
49    return bucket.hasTokens();
50  }
51
52  /**
53   * Get current status of rate limiter
54   * @param key - Endpoint identifier
55   * @returns RateLimitStatus
56   */
57  getStatus(key: string): RateLimitStatus {
58    const bucket = this.buckets.get(key);
59    if (!bucket) {
60      return {
61        configured: false,
62        available: Infinity,
63        resetTime: null,
64      };
65    }
66
67    return bucket.getStatus();
68  }
69
70  /**
71   * Reset rate limiter for specific endpoint
72   * @param key - Endpoint identifier
73   */
74  reset(key: string): void {
75    const bucket = this.buckets.get(key);

```

```

76     if (bucket) {
77         bucket.reset();
78         this.logger.info('Rate limiter reset', { key });
79     }
80 }
81 }
82
83 /**
84  * Token Bucket Implementation
85  */
86 class TokenBucket {
87     private tokens: number;
88     private lastRefill: number;
89     private readonly config: RateLimitConfig;
90     private readonly logger: Logger;
91
92     constructor(config: RateLimitConfig, logger: Logger) {
93         this.config = config;
94         this.logger = logger;
95         this.tokens = config.maxTokens;
96         this.lastRefill = Date.now();
97     }
98
99     /**
100     * Consume a token, waiting if necessary
101     */
102     async consume(): Promise<void> {
103         this.refill();
104
105         if (this.tokens >= 1) {
106             this.tokens -= 1;
107             return;
108         }
109
110         // Calculate wait time
111         const tokensNeeded = 1 - this.tokens;
112         const waitTime = (tokensNeeded / this.config.refillRate) * 1000;
113
114         this.logger.debug('Rate limit hit, waiting', {
115             waitTime: Math.round(waitTime),
116             endpoint: this.config.name
117         });
118
119         await new Promise(resolve => setTimeout(resolve, waitTime));
120
121         // Try again after waiting
122         return this.consume();
123     }
124
125     /**
126     * Check if tokens are available
127     */
128     hasTokens(): boolean {
129         this.refill();
130         return this.tokens >= 1;
131     }
132
133     /**
134     * Get current status
135     */
136     getStatus(): RateLimitStatus {
137         this.refill();
138

```

```

139     const nextTokenTime = this.tokens < this.config.maxTokens
140       ? Date.now() + ((1 - (this.tokens % 1)) / this.config.refillRate) * 1000
141       : null;
142
143     return {
144       configured: true,
145       available: Math.floor(this.tokens),
146       resetTime: nextTokenTime,
147       maxTokens: this.config.maxTokens,
148       refillRate: this.config.refillRate,
149     };
150   }
151
152   /**
153    * Reset bucket to full capacity
154    */
155   reset(): void {
156     this.tokens = this.config.maxTokens;
157     this.lastRefill = Date.now();
158   }
159
160   /**
161    * Refill tokens based on elapsed time
162    */
163   private refill(): void {
164     const now = Date.now();
165     const elapsed = (now - this.lastRefill) / 1000; // Convert to seconds
166     const tokensToAdd = elapsed * this.config.refillRate;
167
168     this.tokens = Math.min(this.config.maxTokens, this.tokens + tokensToAdd);
169     this.lastRefill = now;
170   }
171 }
172
173 /**
174  * Cache Manager - Multi-layer Caching System
175  */
176 export class CacheManager {
177   private memoryCache = new Map<string, CacheEntry>();
178   private readonly logger: Logger;
179   private cleanupInterval: NodeJS.Timeout;
180
181   constructor(logger: Logger) {
182     this.logger = logger;
183
184     // Cleanup expired entries every 5 minutes
185     this.cleanupInterval = setInterval(() => {
186       this.cleanup();
187     }, 300000);
188   }
189
190   /**
191    * Get value from cache
192    * @param key - Cache key
193    * @returns Promise<T | null>
194    */
195   async get<T>(key: string): Promise<T | null> {
196     const entry = this.memoryCache.get(key);
197
198     if (!entry) {
199       return null;
200     }
201

```

```

202     if (this.isExpired(entry)) {
203         this.memoryCache.delete(key);
204         return null;
205     }
206
207     // Update access time for LRU
208     entry.lastAccessed = Date.now();
209
210     this.logger.debug('Cache hit', { key });
211     return entry.value as T;
212 }
213
214 /**
215  * Set value in cache
216  * @param key - Cache key
217  * @param value - Value to cache
218  * @param ttl - Time to live in milliseconds
219  */
220 async set<T>(key: string, value: T, ttl: number): Promise<void> {
221     const entry: CacheEntry = {
222         value,
223         expiresAt: Date.now() + ttl,
224         createdAt: Date.now(),
225         lastAccessed: Date.now(),
226         size: this.estimateSize(value),
227     };
228
229     this.memoryCache.set(key, entry);
230
231     this.logger.debug('Cache set', { key, ttl, size: entry.size });
232
233     // Check if we need to evict entries
234     this.evictIfNeeded();
235 }
236
237 /**
238  * Delete value from cache
239  * @param key - Cache key
240  */
241 async delete(key: string): Promise<void> {
242     const deleted = this.memoryCache.delete(key);
243     if (deleted) {
244         this.logger.debug('Cache delete', { key });
245     }
246 }
247
248 /**
249  * Clear all cache entries
250  */
251 async clear(): Promise<void> {
252     const size = this.memoryCache.size;
253     this.memoryCache.clear();
254     this.logger.info('Cache cleared', { entriesRemoved: size });
255 }
256
257 /**
258  * Get cache statistics
259  */
260 getStats(): CacheStats {
261     let totalSize = 0;
262     let expiredCount = 0;
263     const now = Date.now();
264

```

```

265     for (const entry of this.memoryCache.values()) {
266         totalSize += entry.size;
267         if (this.isExpired(entry)) {
268             expiredCount++;
269         }
270     }
271
272     return {
273         totalEntries: this.memoryCache.size,
274         totalSize,
275         expiredEntries: expiredCount,
276         hitRate: 0, // Would need to track hits/misses for this
277     };
278 }
279
280 /**
281  * Cleanup expired entries
282  */
283 private cleanup(): void {
284     const before = this.memoryCache.size;
285     const now = Date.now();
286
287     for (const [key, entry] of this.memoryCache.entries()) {
288         if (this.isExpired(entry)) {
289             this.memoryCache.delete(key);
290         }
291     }
292
293     const removed = before - this.memoryCache.size;
294     if (removed > 0) {
295         this.logger.debug('Cache cleanup completed', { entriesRemoved: removed });
296     }
297 }
298
299 /**
300  * Evict entries if cache is too large
301  */
302 private evictIfNeeded(): void {
303     const maxEntries = 1000; // Maximum number of entries
304     const maxSize = 50 * 1024 * 1024; // 50MB
305
306     if (this.memoryCache.size <= maxEntries) {
307         return;
308     }
309
310     // Convert to array and sort by last accessed time (LRU)
311     const entries = Array.from(this.memoryCache.entries())
312         .sort(([, a], [, b]) => a.lastAccessed - b.lastAccessed);
313
314     // Remove oldest entries until we're under the limit
315     const toRemove = this.memoryCache.size - maxEntries;
316     for (let i = 0; i < toRemove; i++) {
317         const [key] = entries[i];
318         this.memoryCache.delete(key);
319     }
320
321     this.logger.debug('Cache eviction completed', { entriesRemoved: toRemove });
322 }
323
324 /**
325  * Check if cache entry is expired
326  */
327 private isExpired(entry: CacheEntry): boolean {

```

```
328     return Date.now() > entry.expiresAt;
329 }
330
331 /**
332  * Estimate size of cached value
333  */
334 private estimateSize(value: any): number {
335     try {
336         return JSON.stringify(value).length * 2; // Rough estimate (UTF-16)
337     } catch {
338         return 1000; // Default size if serialization fails
339     }
340 }
341
342 /**
343  * Cleanup on destruction
344  */
345 destroy(): void {
346     if (this.cleanupInterval) {
347         clearInterval(this.cleanupInterval);
348     }
349     this.memoryCache.clear();
350 }
351 }
```

Listing 5: Sophisticated Rate Limiting System

6 Conclusion

This comprehensive API integration guide provides detailed implementation patterns for all external services used in the Jupiter Swap DApp. The integration follows best practices for reliability, performance, and maintainability.

6.1 Integration Summary

Key Integration Features:

- **Jupiter API v6:** Complete integration with advanced features
- **Multi-RPC Strategy:** Helius primary, Alchemy backup
- **Market Data:** CoinGecko integration for optimization
- **Rate Limiting:** Sophisticated token bucket implementation
- **Caching:** Multi-layer caching with LRU eviction
- **Error Handling:** Comprehensive retry and fallback logic
- **Performance:** Optimized for speed and reliability
- **Monitoring:** Detailed logging and metrics

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