TAMBO MCP Integration Suite - API Documentation

🔌 API Overview

This guide covers all API integrations within the TAMBO MCP Integration Suite, including ABACUS Intelligence, TAMBO Components API, and MCP Protocol endpoints.

🧠 ABACUS Intelligence API

Base Configuration

```
const config = {
  appId: '1573da0c2c',
  baseUrl: 'https://apps.abacus.ai/chatllm',
  enableRealTimeUpdates: true
}
```

1. Route Request API

Purpose: Intelligent routing of user requests to appropriate agents

```
async routeRequest(tier: string, payload: string, environment?: string): Promise<Aba-
cusResponse>
```

Parameters:
- `tier` (string): User tier - “Standard”, “Pro”, “Enterprise”
- `payload` (string): User request content
- `environment` (string, optional): “development” | “production”

Response Format:

```
interface AbacusResponse {
  agent: string;              // Chosen agent name
  tier: string;               // User tier
  intent: string;             // Detected intent
  route: string;              // MCP endpoint
  notes: string;              // Decision explanation
  timestamp: string;          // ISO timestamp
  confidence: number;         // Confidence score (0-1)
  metadata: {
    keywords_matched: string[];
    processing_time_ms: number;
  };
}
```

Example Usage:

```
const response = await abacusClient.routeRequest('Pro', 'I need help with a login error');

// Response:
{
  "agent": "TriageAgent",
  "tier": "Pro",
  "intent": "support_request",
  "route": "/triage",
  "notes": "Detected error, issue related request; routed to TriageAgent.",
  "timestamp": "2025-08-07T10:30:00.000Z",
  "confidence": 0.95,
  "metadata": {
    "keywords_matched": ["error", "help"],
    "processing_time_ms": 127
  }
}
```

2. Component Update API

Purpose: AI-powered component modification analysis

```
async componentUpdate(componentId: string, updateInstructions: string, author: string, environment?: string): Promise<ComponentUpdateResponse>
```

Parameters:

- `componentId` (string): Target component identifier
- `updateInstructions` (string): Natural language update description
- `author` (string): User making the change
- `environment` (string, optional): Deployment environment

Example Usage:

```
const response = await abacusClient.componentUpdate(
  'nav-header',
  'Make the navigation more mobile-friendly with hamburger menu',
  'admin@tambo.ai'
);
```

3. Agent Diagnostics API

Purpose: Analyze agent performance and health

```
async agentDiagnostics(agent: string, scope: string, environment?: string): Promise<DiagnosticsResponse>
```

Parameters:

- `agent` (string): Agent name to analyze
- `scope` (string): “fast” | “detailed” | “comprehensive”
- `environment` (string, optional): Target environment

🏗️ TAMBO Components API

Base Configuration

```
const config = {
  apiKey: 'tambo_2crvFKf2vvsK8WYmBToavxmgJF+jeuR0o5yNaNUBxhP1L56c6YeCZao0/voar1-
gR47s4yevBC0QQ/XfIfBE9aAueUIBiHEosmPHJv4JVjqY=',
  baseUrl: 'https://api.tambo.co/v1',
  safeMode: true,
  maxRequestsPerMinute: 30
}
```

1. Get Component

Endpoint: `GET /components/{componentId}`

```
async getComponent(componentId: string): Promise<TamboComponent>
```

Response Format:

```
interface TamboComponent {
  id: string;
  name: string;
  code: string;
  props: any;
  category: string;
  tags: string[];
}
```

Example:

```
const component = await tamboClient.getComponent('header-nav');
// Returns complete component definition with code and metadata
```

2. Update Component

Endpoint: `PATCH /components/{componentId}`

```
async updateComponent(componentId: string, changes: any): Promise<TamboComponent>
```

Safety Guards Applied:
- Protected component validation
- Dangerous code pattern detection
- Rate limit enforcement

Example:

```
const changes = {
  style: { backgroundColor: 'blue' },
  props: { title: 'New Title' }
};

const updated = await tamboClient.updateComponent('my-button', changes);
```

3. Search Components

Endpoint: `GET /components/search?q={query}`

```
async searchComponents(query: string): Promise<TamboComponent[]>
```

Example:

```
const results = await tamboClient.searchComponents('navigation button');
// Returns array of matching components
```

4. Safety Controls

Enable/Disable Safe Mode:

```
tamboClient.enableSafeMode();   // Protect core components
tamboClient.disableSafeMode();  // Allow all modifications (caution!)
```

Check Status:

```
const status = tamboClient.getStatus();
// Returns: { safeMode: boolean, requestsRemaining: number, protectedComponents: num-
ber }
```

🔄 MCP Integration Endpoints

1. Command Processing

```
async processCommand(command: string, context?: CommandContext): Promise<CommandRe-
sponse>
```

Context Parameters:

```
interface CommandContext {
  abacusEnhanced: boolean;
  timestamp: string;
  environment: string;
  validationOnly?: boolean;
  requirements?: any;
}
```

Example Commands:

```
// Component modification
await commandEngine.processCommand('Modify button-primary: make it larger and blue');

// Component creation
await commandEngine.processCommand('Create a responsive card component with image and
text');

// Validation only
await commandEngine.processCommand('Validate: add dark mode to sidebar', { valida-
tionOnly: true });
```

2. Cross-Tool Search

```
async searchUnified(query: string, filters?: SearchFilter): Promise<CrossToolResult[]>
```

Search Filters:

```
interface SearchFilter {
  sources: string[];        // ['tambo', 'figma', 'github']
  types: string[];          // ['component', 'design', 'code']
  dateRange?: DateRange;
  categories?: string[];
}
```

Example:

```
const results = await crossToolSearch.searchUnified('button components', {
  sources: ['tambo', 'figma'],
  types: ['component', 'design']
});
```

3. Sync Operations

```
// Sync single component
await mcpSync.syncComponent('component-id');

// Sync all components
await mcpSync.syncAllComponents();

// Get sync status
const status = await mcpSync.getSyncStatus();
```

🎯 Routing Logic Reference

Agent Assignment Rules

Keywords	Agent	Route	Description
support, error, issue, bug, problem	TriageAgent	/triage	Technical support and troubleshooting
blog, article, media, content, post	ContentRouterAgent	/content	Content management and publishing
feedback, comment, review, rating	FeedbackMinerAgent	/feedback	User feedback analysis
pricing, upgrade, tier, billing, payment	PricingIntelligenceAgent	/pricing	Pricing and subscription management
log, record, compliance, audit, track	AuditAgent	/audit	Logging and compliance tracking

Tier-Based Features

Feature	Standard	Pro	Enterprise
Route Requests	✅	✅	✅
Component Updates	❌	✅	✅
Agent Diagnostics	❌	Limited	Full
Cross-Tool Search	❌	✅	✅
Real-time Sync	❌	❌	✅

📊 Rate Limits & Quotas

TAMBO API Limits

- **Requests**: 30 per minute per API key
- **Component Updates**: 10 per hour in safe mode
- **Search Queries**: 100 per hour

ABACUS Integration Limits

- **Route Requests**: Unlimited (chatbot-based)
- **Processing Time**: ~800ms average
- **Context Storage**: 100 conversations

MCP Protocol Limits

- **Sync Operations**: 1 per 30 seconds

- **Command Processing**: 50 per hour
- **Cross-tool Searches**: 20 per hour

🔐 Authentication & Security

API Key Management

```
// Environment-based configuration
const apiKey = process.env.TAMBO_API_KEY || 'fallback-dev-key';

// Request headers
const headers = {
  'Authorization': `Bearer ${apiKey}`,
  'Content-Type': 'application/json',
  'User-Agent': 'TAMBO-MCP-Suite/1.0.0'
};
```

Error Handling Patterns

```
try {
  const result = await tamboClient.updateComponent(id, changes);
} catch (error) {
  if (error.message.includes('Rate limit')) {
    // Handle rate limiting
    await this.waitAndRetry();
  } else if (error.message.includes('protected')) {
    // Handle protected component error
    this.showProtectedComponentWarning();
  } else {
    // Generic error handling
    this.logError(error);
    throw error;
  }
}
```

Response Status Codes

Code	Meaning	Action
200	Success	Process response
400	Bad Request	Validate input parameters
401	Unauthorized	Check API key
403	Forbidden	Verify permissions
429	Rate Limited	Implement backoff
500	Server Error	Retry with exponential back-off

🧪 Testing APIs

Mock Data Examples

```
// Mock ABACUS response
const mockRoutingResponse = {
  agent: 'TriageAgent',
  tier: 'Pro',
  intent: 'support_request',
  route: '/triage',
  notes: 'Test routing response',
  timestamp: '2025-08-07T10:30:00.000Z',
  confidence: 0.95,
  metadata: {
    keywords_matched: ['support'],
    processing_time_ms: 125
  }
};

// Mock component data
const mockComponent = {
  id: 'test-button',
  name: 'Test Button',
  code: 'export const TestButton = () => <button>Test</button>',
  props: { variant: 'primary' },
  category: 'forms',
  tags: ['button', 'interactive']
};
```

Integration Testing

```
describe('API Integration Tests', () => {
  test('ABACUS routing', async () => {
    const result = await abacusClient.routeRequest('Pro', 'support request');
    expect(result.agent).toBeDefined();
    expect(result.confidence).toBeGreaterThan(0.5);
  });

  test('TAMBO component update', async () => {
    const changes = { color: 'blue' };
    const result = await tamboClient.updateComponent('test-component', changes);
    expect(result.id).toBe('test-component');
  });
});
```

📈 Monitoring & Analytics

Request Logging

```
// Automatic request logging
const logRequest = (endpoint: string, method: string, duration: number) => {
  console.log(`[API] ${method} ${endpoint} - ${duration}ms`);

  // Send to analytics service
  analytics.track('api_request', {
    endpoint,
    method,
    duration,
    timestamp: new Date().toISOString()
  });
};
```

Performance Metrics

- **Average Response Time**: Track per endpoint
- **Success Rate**: Monitor error rates
- **Rate Limit Usage**: Track quota consumption
- **Cache Hit Rate**: Monitor caching effectiveness

For troubleshooting API issues, see the Troubleshooting Guide (05_TROUBLESHOOTING.md)