# TAMBO MCP Integration Suite - Technical Implementation Guide

## 🏗️ Architecture Deep Dive

### Frontend Stack

- **Framework**: React 18 with TypeScript
- **Build Tool**: Vite for fast development
- **UI Components**: Radix UI + Tailwind CSS
- **State Management**: React Hooks + Custom hooks
- **HTTP Client**: Axios for API calls

### Backend Integration

- **TAMBO API**: RESTful component management
- **ABACUS Intelligence**: Chatbot-powered routing
- **MCP Protocol**: Cross-tool communication
- **Real-time Sync**: WebSocket connections (planned)

## 🔌 API Integration Details

### ABACUS Client Implementation

```
// Core routing intelligence
async routeRequest(tier: string, payload: string, environment = 'development'):
Promise<AbacusResponse> {
  const prompt = this.buildRoutingPrompt(tier, payload, environment);
  const response = await this.sendToAbacus(prompt);
  return this.parseRoutingResponse(response);
}
```

**Key Features:**
- Intelligent prompt construction
- JSON response parsing
- Conversation history tracking
- Error handling with fallbacks

### TAMBO Client Integration

```
// Safe component modification
async updateComponent(componentId: string, changes: any): Promise<TamboComponent> {
  // Safety validation
  if (!this.safetyGuards.canModifyComponent(componentId)) {
    throw new Error(`Component ${componentId} is protected`);
  }

  if (!this.safetyGuards.validateChange(changes)) {
    throw new Error('Dangerous code detected');
  }

  // Execute API call
  return await this.makeRequest(`/components/${componentId}`, {
    method: 'PATCH',
    body: JSON.stringify({ changes, safeMode: true })
  });
}
```

**Safety Features:**
- Protected component validation
- Dangerous code pattern detection
- Rate limiting enforcement
- Request logging and monitoring

## 🧠 MCP Integration Layer

### Command Engine Architecture

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

class CommandEngine {
  async processCommand(command: string, context?: CommandContext): Promise<any> {
    // Parse natural language
    const parsed = this.parseNaturalLanguage(command);

    // Route to appropriate handler
    return await this.routeCommand(parsed, context);
  }
}
```

### Cross-Tool Search Implementation

```
interface SearchFilter {
  sources: string[];
  types: string[];
  dateRange?: DateRange;
  categories?: string[];
}

class CrossToolSearch {
  async searchUnified(query: string, filters?: SearchFilter):
Promise<CrossToolResult[]> {
    // Multi-source search execution
    const sources = await Promise.allSettled([
      this.searchTambo(query, filters),
      this.searchMCP(query, filters),
      this.searchExternal(query, filters)
    ]);

    // Merge and rank results
    return this.mergeResults(sources);
  }
}
```

## 🔄 State Management

### Custom Hooks Pattern

```
// Main MCP integration hook
export const useMCPIntegration = (options: MCPHookOptions) => {
  const [state, setState] = useState<MCPState>({
    isLoading: false,
    lastCommand: '',
    results: null,
    error: null
  });

  // Enhanced command processing with ABACUS
  const sendCommand = useCallback(async (command: string, context?: any) => {
    // Route through ABACUS for intelligence
    if (command.includes('route')) {
      return await abacusClient.routeRequest(tier, payload, environment);
    }

    // Fallback to traditional processing
    return await commandEngine.processCommand(command, context);
  }, []);

  return { state, sendCommand, /* other methods */ };
};
```

### Specialized Hooks

1. **useTamboComponents**: Component-specific operations
2. **useMCPSync**: Real-time synchronization
3. **useTamboLive**: Live data updates

## 🛡️ Security Implementation

### API Key Management

```
class TamboAPIClient {
  private readonly API_KEY = 'tambo_2crvFKf2vvsK8WYmBToavxmgJF+jeuR0o5yNaNUBxhP1L56c6Ye
CZao0/voar1gR47s4yevBC0QQ/XfIfBE9aAueUIBiHEosmPHJv4JVjqY=';

  constructor() {
    // Key validation and encryption in production
    this.validateApiKey();
  }
}
```

### Safety Guards Implementation

```
const safetyGuards: SafetyGuards = {
  canModifyComponent: (componentId: string) => {
    return !PROTECTED_COMPONENTS.includes(componentId) || !safeMode;
  },

  validateChange: (change: any) => {
    const dangerousPatterns = [
      'dangerouslySetInnerHTML',
      'eval(',
      'Function(',
      // ... more patterns
    ];

    const changeStr = JSON.stringify(change);
    return !dangerousPatterns.some(pattern => changeStr.includes(pattern));
  },

  rateLimitCheck: () => {
    return this.requestCount < this.maxRequestsPerMinute;
  }
};
```

## 📡 Real-time Features

### WebSocket Integration (Planned)

```
class MCPSync {
  private ws: WebSocket;

  async initializeSync() {
    this.ws = new WebSocket('wss://api.tambo.co/v1/sync');

    this.ws.onmessage = (event) => {
      const data = JSON.parse(event.data);
      this.handleSyncUpdate(data);
    };
  }

  async syncAllComponents() {
    // Batch sync logic
    const components = await this.getAllComponents();
    const syncPromises = components.map(c => this.syncComponent(c.id));
    await Promise.allSettled(syncPromises);
  }
}
```

## 🧪 Testing Strategy

### Unit Testing

```
// Component testing
describe('TamboRoutingConsolePro', () => {
  test('should route request correctly', async () => {
    const result = await abacusClient.routeRequest('Pro', 'support request');
    expect(result.agent).toBe('TriageAgent');
    expect(result.route).toBe('/triage');
  });
});
```

### Integration Testing

```
// API integration testing
describe('TAMBO API Integration', () => {
  test('should update component safely', async () => {
    const changes = { color: 'blue' };
    const result = await tamboClient.updateComponent('test-component', changes);
    expect(result.id).toBe('test-component');
  });
});
```

## 🔍 Error Handling

### Graceful Degradation

```
class AbacusAPIClient {
  async routeRequest(tier: string, payload: string): Promise<AbacusResponse> {
    try {
      return await this.callAbacusAPI(tier, payload);
    } catch (error) {
      console.error('ABACUS failed, using fallback routing');
      return this.fallbackRouting(tier, payload);
    }
  }

  private fallbackRouting(tier: string, payload: string): AbacusResponse {
    // Simple keyword-based routing as fallback
    return this.keywordBasedRouting(payload);
  }
}
```

### Error Boundaries

```
// React error boundary for graceful UI failures
class MCPErrorBoundary extends React.Component {
  componentDidCatch(error: Error, errorInfo: React.ErrorInfo) {
    console.error('MCP Integration Error:', error, errorInfo);
    this.setState({ hasError: true, error });
  }
}
```

## 📈 Performance Optimization

### Caching Strategy

```
class ComponentCache {
  private cache = new Map<string, { data: any, timestamp: number }>();
  private readonly TTL = 5 * 60 * 1000; // 5 minutes

  async get(componentId: string): Promise<any> {
    const cached = this.cache.get(componentId);
    if (cached && Date.now() - cached.timestamp < this.TTL) {
      return cached.data;
    }

    // Fetch fresh data
    const fresh = await tamboClient.getComponent(componentId);
    this.cache.set(componentId, { data: fresh, timestamp: Date.now() });
    return fresh;
  }
}
```

### Request Batching

```
class RequestBatcher {
  private queue: Array<{ request: Promise<any>, resolve: Function, reject: Function }>
= [];

  async batchRequest<T>(request: () => Promise<T>): Promise<T> {
    return new Promise((resolve, reject) => {
      this.queue.push({ request: request(), resolve, reject });

      // Process queue after short delay
      setTimeout(() => this.processQueue(), 100);
    });
  }
}
```

## 🔧 Development Setup

### Environment Configuration

```
// Environment-specific configurations
const config = {
  development: {
    tamboApiUrl: 'https://api-dev.tambo.co/v1',
    abacusAppId: 'dev-1573da0c2c',
    enableDebugLogs: true,
    mockModeEnabled: true
  },
  production: {
    tamboApiUrl: 'https://api.tambo.co/v1',
    abacusAppId: '1573da0c2c',
    enableDebugLogs: false,
    mockModeEnabled: false
  }
};
```

### Build Configuration

```
// vite.config.ts
export default defineConfig({
  plugins: [react()],
  define: {
    __TAMBO_API_KEY__: JSON.stringify(process.env.TAMBO_API_KEY),
    __ABACUS_APP_ID__: JSON.stringify(process.env.ABACUS_APP_ID)
  },
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          'mcp-core': ['./src/mcp/tambo-integration.ts'],
          'services': ['./src/services/abacusClient.ts', './src/services/tamboCli-
ent.ts']
        }
      }
    }
  }
});
```

## 📋 Code Standards

### TypeScript Best Practices

- Strict type checking enabled
- Interface-first design
- Generic types for reusability
- Proper error type definitions

### Code Organization

```
src/
├── components/         # React components
├── hooks/              # Custom React hooks
├── services/           # API clients
├── mcp/                # MCP integration layer
├── types/              # TypeScript definitions
└── utils/              # Utility functions
```

### Testing Structure

```
__tests__/
├── components/        # Component tests
├── services/          # Service tests
├── hooks/             # Hook tests
└── integration/       # Integration tests
```

---

For practical examples and usage patterns, see the User Guide (04_USER_GUIDE.md)