TAMBO MCP Integration Suite - Troubleshooting Guide

🚨 Common Issues & Solutions

1. API Connection Issues

ABACUS Intelligence Connection Failed

Symptoms:
- “ABACUS Intelligence Error” messages
- Routing requests timing out
- Fallback routing being used

Diagnosis:

```
// Check ABACUS configuration
console.log('ABACUS Config:', abacusClient.getConfig());

// Test connection
try {
  const test = await abacusClient.routeRequest('Pro', 'test connection');
  console.log('ABACUS is working:', test);
} catch (error) {
  console.error('ABACUS failed:', error.message);
}
```

Solutions:
1. **Check App ID**: Verify `appId: '1573da0c2c'` is correct
2. **Network Issues**: Test direct access to `https://apps.abacus.ai/chatllm`
3. **Clear Cache**: Reset conversation history
`typescript`
`  abacusClient.clearHistory();`
4. **Fallback Mode**: System automatically uses keyword-based routing

TAMBO API Connection Issues

Symptoms:
- “Tambo API Error: 401” or “403” responses
- Component operations failing
- Rate limit errors

Diagnosis:

```
// Check API key validity
const status = tamboClient.getStatus();
console.log('TAMBO Status:', status);

// Test API connection
try {
  const test = await tamboClient.getComponent('test-component');
  console.log('TAMBO API is working');
} catch (error) {
  console.error('TAMBO API Error:', error.message);
}
```

Solutions:

1. **API Key Issues**:

`typescript`
`  // Verify API key format`
`  const apiKey = 'tambo_2crvFKf2vvsK8WYmBToavxmgJF+jeuR0o5yNaNUBxhP1L56c6YeCZao0/voar1-gR47s4yevBC0QQ/XfIfBE9aAueUIBiHEosmPHJv4JVjqY=';`
`  console.log('API Key valid format:', apiKey.startsWith('tambo_'));`

1. **Rate Limit Exceeded**:

``typescript``
`  // Check rate limit status`
`  const status = tamboClient.getStatus();`
`  console.log( Requests remaining: ${status.requestsRemaining}/30`);

// Wait and retry
if (status.requestsRemaining === 0) {
setTimeout(() => {
// Retry your request
}, 60000); // Wait 1 minute
}
```

1. **Network/CORS Issues**:
   - Check if `https://api.tambo.co` is accessible
   - Verify CORS settings in development
   - Test with different network connection

## 2. Component Operation Errors

### Protected Component Modification

**Error Message**: `"Component [id] is protected and cannot be modified"`

**Cause**: Attempting to modify core system components

**Protected Components List**:
- `core-layout`
- `main-navigation`
- `app-shell`
- `error-boundary`

**Solutions**:
1. **Check Component ID**:
```

```typescript
const PROTECTED_COMPONENTS = [
'core-layout', 'main-navigation', 'app-shell', 'error-boundary'
];

if (PROTECTED_COMPONENTS.includes(componentId)) {
console.warn( ⚠️ `${componentId} is protected` );
// Use alternative approach
}
```

1. **Admin Override** (CEO/Admin only):
   `typescript`
   `tamboClient.disableSafeMode();`
   `// Proceed with caution - can break system`

2. **Alternative Components**:
   - Instead of `main-navigation` → modify `nav-items`
   - Instead of `core-layout` → modify `page-content`

Dangerous Code Detection

Error Message: `"Change validation failed - potentially dangerous code detected"`

Dangerous Patterns:

```
const dangerousPatterns = [
  'dangerouslySetInnerHTML',
  'eval(',
  'Function(',
  'setTimeout(',
  'setInterval(',
  'document.cookie',
  'localStorage.clear()',
  'window.location'
];
```

Solutions:
1. **Review Your Changes**:
```typescript
const changes = {
// ❌ Dangerous
innerHTML: '',
```

```
  // ✅ Safe
  textContent: 'Safe text content',
  style: { color: 'blue' }
```

};
```

1. **Use Safe Alternatives**:
   ```typescript

```
    // Instead of eval()
    const result = JSON.parse(safeJsonString);

// Instead of dangerouslySetInnerHTML
const element = {safeTextContent};
```

3. MCP Integration Issues

Sync Failures

Symptoms:
- Components not updating across tools
- Sync status showing errors
- Data inconsistencies

Diagnosis:

```
// Check sync status
const syncStatus = await mcpSync.getSyncStatus();
console.log('Sync Status:', syncStatus);

// Check queue size
console.log('Sync Queue Size:', syncStatus.queueSize);
```

Solutions:
1. **Manual Force Sync**:
`typescript`
`  try {`
`    await mcpSync.syncAllComponents();`
`    console.log('✅ Manual sync completed');`
`  } catch (error) {`
`    console.error('❌ Manual sync failed:', error);`
`  }`

1. **Component-Specific Sync**:
   `typescript`
   `  await mcpSync.syncComponent('specific-component-id');`

2. **Clear Sync Queue**:
   `typescript`
   `  await mcpSync.clearQueue();`
   `  await mcpSync.reinitialize();`

Cross-Tool Search Not Working

Symptoms:
- Search returns empty results
- Slow search performance
- Tool-specific results missing

Solutions:
1. **Check Search Filters**:
`typescript`
`  const filters = {`

```
    sources: ['tambo', 'figma', 'github'], // Ensure all sources enabled
    types: ['component', 'design', 'code'],
    dateRange: undefined // Remove date restrictions
  };
```

1. **Test Individual Sources**:
   ```typescript
   // Test each source separately
   const tamboResults = await crossToolSearch.searchTambo(query);
   const mcpResults = await crossToolSearch.searchMCP(query);
   ```

4. Performance Issues

Slow Response Times

Symptoms:
- Requests taking >5 seconds
- UI freezing during operations
- Timeout errors

Diagnosis:

```
// Measure response times
const start = performance.now();
try {
  const result = await abacusClient.routeRequest('Pro', 'test');
  const duration = performance.now() - start;
  console.log(`Response time: ${duration}ms`);
} catch (error) {
  console.error('Timeout or error:', error);
}
```

Solutions:
1. **Enable Caching**:
```typescript
// Implement request caching
const cache = new Map();

async function cachedRequest(key, requestFn) {
if (cache.has(key)) {
return cache.get(key);
}
```

```
  const result = await requestFn();
  cache.set(key, result);
  setTimeout(() => cache.delete(key), 300000); // 5min TTL
  return result;
```

```
}
```

1. **Request Batching**:
   ```typescript
   // Batch multiple requests
   ```

```
const requests = [
() => tamboClient.getComponent('comp1'),
() => tamboClient.getComponent('comp2'),
() => tamboClient.getComponent('comp3')
];

const results = await Promise.allSettled(requests.map(req => req()));
```

1. **Reduce Payload Size**:
```typescript
// Minimize request data
const minimalRequest = {
  tool: 'routeRequest',
  args: { tier, payload: payload.slice(0, 200) } // Truncate long payloads
};
```

Memory Leaks

Symptoms:
- Browser tab using excessive memory
- Performance degrading over time
- Browser becoming unresponsive

Solutions:
1. **Clear Conversation History**:
```typescript
// Regularly clear history
setInterval(() => {
  if (abacusClient.getConversationHistory().length > 100) {
    abacusClient.clearHistory();
  }
}, 600000); // Every 10 minutes
```

1. **Component Cleanup**:
```typescript
useEffect(() => {
  // Cleanup on unmount
  return () => {
    mcpSync.disconnect();
    abacusClient.clearHistory();
  };
}, []);
```

5. Authentication & Authorization

API Key Validation Errors

Error Messages:
- "Invalid API key format"
- "Unauthorized access"
- "API key expired"

Solutions:
1. **Validate Key Format**:

```typescript
const isValidTamboKey = (key) => {
return key && key.startsWith('tambo_') && key.length > 50;
};

if (!isValidTamboKey(process.env.TAMBO_API_KEY)) {
console.error('❌ Invalid TAMBO API key format');
}
```

1. **Environment Configuration**:
   ```typescript
   // Check environment variables
   const requiredVars = ['TAMBO_API_KEY', 'ABACUS_APP_ID'];
   const missing = requiredVars.filter(v => !process.env[v]);

if (missing.length > 0) {
console.error('❌ Missing environment variables:', missing);
}
```

1. **Key Rotation**:
   ```typescript
   // Implement key rotation for production
   const getApiKey = () => {
   const primaryKey = process.env.TAMBO_API_KEY;
   const fallbackKey = process.env.TAMBO_API_KEY_BACKUP;

   return primaryKey || fallbackKey;
   };
   ```

6. UI/Frontend Issues

Component Not Rendering

Symptoms:
- Blank screen or components
- Console errors about missing components
- Infinite loading states

Diagnosis:

```
// Check React error boundaries
const ErrorBoundary = ({ children }) => {
  const [hasError, setHasError] = useState(false);

  useEffect(() => {
    const errorHandler = (error, errorInfo) => {
      console.error('React Error:', error, errorInfo);
      setHasError(true);
    };

    window.addEventListener('error', errorHandler);
    return () => window.removeEventListener('error', errorHandler);
  }, []);

  if (hasError) {
    return <div>Something went wrong. Check console for details.</div>;
  }

  return children;
};
```

Solutions:
1. **Check Component Dependencies**:
```typescript
// Verify all required props are passed
const TamboRoutingConsolePro = ({
requiredProp1,
requiredProp2
}) => {
if (!requiredProp1 || !requiredProp2) {
console.error('❌ Missing required props');
return

Configuration error
;
}
```

```
  // Component logic...
```

```
};
```

1. **Loading State Management**:
   ```typescript
   const [loading, setLoading] = useState(false);
   const [error, setError] = useState(null);

const handleSubmit = async () => {
setLoading(true);
setError(null);
   ```

```
  try {
    const result = await apiCall();
    // Handle success
  } catch (err) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
```

};
```

### State Management Issues

**Symptoms:**
- UI not updating after API calls
- Inconsistent state across components
- Stale data displayed

**Solutions**:
1. **Proper State Updates**:
`typescript`
`  // Use functional updates for complex state`
`  setState(prevState => ({`
`    ...prevState,`
`    results: newResults,`
`    lastUpdated: new Date().toISOString()`
`  }));`

1. **Effect Dependencies**:
   `typescript`
   `  // Ensure proper effect dependencies`
   `  useEffect(() => {`
   `    fetchData();`
   `  }, [dependency1, dependency2]); // Include all dependencies`
```

🔄 System Recovery Procedures

Complete System Reset

```
// Emergency reset procedure
const emergencyReset = async () => {
  console.log('🚨 Starting emergency reset...');

  // 1. Clear all caches
  abacusClient.clearHistory();
  localStorage.clear();

  // 2. Reset API clients
  tamboClient.enableSafeMode();

  // 3. Force sync
  await mcpSync.clearQueue();
  await mcpSync.reinitialize();

  // 4. Reload application
  window.location.reload();

  console.log('✅ Emergency reset completed');
};
```

Gradual Recovery

```
// Step-by-step recovery
const gradualRecovery = async () => {
  // Step 1: Test basic connectivity
  try {
    await fetch('https://api.tambo.co/v1/health');
    console.log('✅ TAMBO API reachable');
  } catch {
    console.log('❌ TAMBO API unreachable');
  }

  // Step 2: Test ABACUS
  try {
    const test = await abacusClient.routeRequest('Pro', 'health check');
    console.log('✅ ABACUS working');
  } catch {
    console.log('❌ ABACUS failing, using fallback');
  }

  // Step 3: Test MCP sync
  try {
    await mcpSync.syncComponent('test-component');
    console.log('✅ MCP sync working');
  } catch {
    console.log('❌ MCP sync failing');
  }
};
```

📞 Support Escalation

Level 1: Self-Service

- Check this troubleshooting guide
- Review error messages in browser console
- Test in development environment
- Verify API credentials

Level 2: Team Support

- Contact team lead or senior developer
- Provide error logs and reproduction steps
- Include browser/environment information
- Share relevant configuration details

Level 3: System Administration

- Contact system administrators
- Report potential security issues
- Request API key rotation
- Escalate service outages

Level 4: Vendor Support

- Contact TAMBO support for API issues
- Contact ABACUS support for intelligence issues
- Provide detailed error logs and request traces
- Include system configuration information

For additional support, refer to the User Guide (04_USER_GUIDE.md) or Technical Documentation (02_TECHNICAL.md)