MCP Server Implementation Template

Purpose: Implement Model Context Protocol (MCP) server in your satellite app
Timeline: 2-4 hours
Prerequisites: Understanding of MCP protocol

What is MCP?

Model Context Protocol (MCP) enables bidirectional communication between Nexus (conductor) and satellite apps.

MCP Components

1. **Tools** - Actions Nexus can invoke in your app
2. **Resources** - Data Nexus can access from your app
3. **Prompts** - Templates for common tasks

Implementation Steps

Step 1: Create MCP Endpoint (30 minutes)

```typescript
// /api/mcp/route.ts
import { NextRequest, NextResponse } from 'next/server';

export async function POST(request: NextRequest) {
  try {
    const { method, params } = await request.json();

    switch (method) {
      case 'initialize':
        return handleInitialize();
      case 'tools/list':
        return handleToolsList();
      case 'tools/call':
        return handleToolsCall(params);
      case 'resources/list':
        return handleResourcesList();
      case 'resources/read':
        return handleResourcesRead(params);
      case 'prompts/list':
        return handlePromptsList();
      case 'prompts/get':
        return handlePromptsGet(params);
      default:
        return NextResponse.json(
          { error: 'Unknown method' },
          { status: 400 }
        );
    }
  } catch (error) {
    console.error('MCP error:', error);
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    );
  }
}

function handleInitialize() {
  return NextResponse.json({
    protocolVersion: '2024-11-05',
    capabilities: {
      tools: {},
      resources: {},
      prompts: {}
    },
    serverInfo: {
      name: process.env.APP_ID || 'satellite-app',
      version: process.env.APP_VERSION || '1.0.0'
    }
  });
}
```

Step 2: Define Tools (1 hour)

Tools are actions Nexus can invoke in your app.

```
function handleToolsList() {
  return NextResponse.json({
    tools: [
      {
        name: 'get_data',
        description: 'Retrieve data from this application',
        inputSchema: {
          type: 'object',
          properties: {
            query: {
              type: 'string',
              description: 'Search query'
            },
            limit: {
              type: 'number',
              description: 'Maximum number of results',
              default: 10
            }
          },
          required: ['query']
        }
      },
      {
        name: 'create_record',
        description: 'Create a new record in this application',
        inputSchema: {
          type: 'object',
          properties: {
            data: {
              type: 'object',
              description: 'Record data'
            }
          },
          required: ['data']
        }
      },
      {
        name: 'update_record',
        description: 'Update an existing record',
        inputSchema: {
          type: 'object',
          properties: {
            id: {
              type: 'string',
              description: 'Record ID'
            },
            data: {
              type: 'object',
              description: 'Updated data'
            }
          },
          required: ['id', 'data']
        }
      }
    ]
  });
}
```

```
async function handleToolsCall(params: any) {
  const { name, arguments: args } = params;

  switch (name) {
    case 'get_data':
      const data = await getData(args.query, args.limit);
      return NextResponse.json({
        content: [
          {
            type: 'text',
            text: JSON.stringify(data, null, 2)
          }
        ]
      });

    case 'create_record':
      const newRecord = await createRecord(args.data);
      return NextResponse.json({
        content: [
          {
            type: 'text',
            text: `Record created: ${newRecord.id}`
          }
        ]
      });

    case 'update_record':
      const updatedRecord = await updateRecord(args.id, args.data);
      return NextResponse.json({
        content: [
          {
            type: 'text',
            text: `Record updated: ${updatedRecord.id}`
          }
        ]
      });

    default:
      return NextResponse.json(
        { error: 'Unknown tool' },
        { status: 400 }
      );
  }
}

// Implement your actual data functions
async function getData(query: string, limit: number) {
  // Your implementation here
  return [];
}

async function createRecord(data: any) {
  // Your implementation here
  return { id: 'new-id' };
}

async function updateRecord(id: string, data: any) {
  // Your implementation here
  return { id };
}
```

Step 3: Define Resources (30 minutes)

Resources are data Nexus can access from your app.

```
function handleResourcesList() {
  return NextResponse.json({
    resources: [
      {
        uri: 'app://data/all',
        name: 'All Data',
        description: 'Complete dataset from this application',
        mimeType: 'application/json'
      },
      {
        uri: 'app://data/recent',
        name: 'Recent Data',
        description: 'Data from the last 7 days',
        mimeType: 'application/json'
      },
      {
        uri: 'app://stats',
        name: 'Statistics',
        description: 'Application statistics and metrics',
        mimeType: 'application/json'
      }
    ]
  });
}

async function handleResourcesRead(params: any) {
  const { uri } = params;

  let data;
  switch (uri) {
    case 'app://data/all':
      data = await getAllData();
      break;
    case 'app://data/recent':
      data = await getRecentData();
      break;
    case 'app://stats':
      data = await getStats();
      break;
    default:
      return NextResponse.json(
        { error: 'Unknown resource' },
        { status: 404 }
      );
  }

  return NextResponse.json({
    contents: [
      {
        uri,
        mimeType: 'application/json',
        text: JSON.stringify(data, null, 2)
      }
    ]
  });
}

// Implement your actual resource functions
```

```
async function getAllData() {
  // Your implementation here
  return [];
}

async function getRecentData() {
  // Your implementation here
  return [];
}

async function getStats() {
  // Your implementation here
  return {};
}
```

Step 4: Define Prompts (30 minutes)

Prompts are templates for common tasks.

```
function handlePromptsList() {
  return NextResponse.json({
    prompts: [
      {
        name: 'analyze_data',
        description: 'Analyze data from this application',
        arguments: [
          {
            name: 'timeframe',
            description: 'Time period to analyze',
            required: false
          }
        ]
      },
      {
        name: 'generate_report',
        description: 'Generate a report',
        arguments: [
          {
            name: 'report_type',
            description: 'Type of report to generate',
            required: true
          }
        ]
      }
    ]
  });
}

async function handlePromptsGet(params: any) {
  const { name, arguments: args } = params;

  switch (name) {
    case 'analyze_data':
      const timeframe = args?.timeframe || 'last 7 days';
      return NextResponse.json({
        messages: [
          {
            role: 'user',
            content: {
              type: 'text',
              text: `Analyze the data from ${process.env.APP_NAME} for ${timeframe}.
                    Identify trends, anomalies, and key insights.`
            }
          }
        ]
      });

    case 'generate_report':
      const reportType = args?.report_type;
      return NextResponse.json({
        messages: [
          {
            role: 'user',
            content: {
              type: 'text',
              text: `Generate a ${reportType} report from ${process.env.APP_NAME}.
                    Include key metrics, visualizations, and recommendations.`
            }
          }
        ]
      });

    default:
      return NextResponse.json(
        { error: 'Unknown prompt' },
        { status: 404 }
      );
  }
}
```

Step 5: Update Registration (10 minutes)

Update your `app-registration.json`:

```json
{
  "technical": {
    "mcp_enabled": true,
    "mcp_endpoint": "https://your-app.abacusai.app/api/mcp"
  }
}
```

Step 6: Test MCP Server (30 minutes)

Test your MCP server:

```bash
# Test initialize
curl -X POST https://your-app.abacusai.app/api/mcp \
  -H "Content-Type: application/json" \
  -d '{"method": "initialize"}'

# Test tools/list
curl -X POST https://your-app.abacusai.app/api/mcp \
  -H "Content-Type: application/json" \
  -d '{"method": "tools/list"}'

# Test tools/call
curl -X POST https://your-app.abacusai.app/api/mcp \
  -H "Content-Type: application/json" \
  -d '{
    "method": "tools/call",
    "params": {
      "name": "get_data",
      "arguments": {"query": "test", "limit": 5}
    }
  }'

# Test resources/list
curl -X POST https://your-app.abacusai.app/api/mcp \
  -H "Content-Type: application/json" \
  -d '{"method": "resources/list"}'
```

Best Practices

Security

- Validate all inputs
- Implement authentication
- Rate limit requests
- Log all MCP calls

Performance

- Cache frequently accessed resources
- Implement pagination for large datasets
- Use async operations
- Monitor response times

Error Handling

- Return clear error messages
- Log errors for debugging
- Handle edge cases gracefully
- Provide fallback responses

Success Criteria

- MCP endpoint responds to all methods
- Tools execute correctly
- Resources return valid data
- Prompts generate useful templates
- Registration updated with MCP endpoint

Reporting

When complete, update your GitHub issue:

```
[APP-ID] MCP Server - Complete

- ☐ MCP endpoint implemented
- ☐ Tools defined and tested
- ☐ Resources exposed
- ☐ Prompts created
- ☐ Registration updated

MCP endpoint: https://your-app.abacusai.app/api/mcp
```