# Developer Assistant MCP Tool: Comprehensive Development Plan

*A complete guide for building a Model Context Protocol tool integrating Google Calendar, Notion, and GitHub APIs using TypeScript, designed for two developers with no prior MCP experience to deliver a working prototype in one afternoon.*

## Project overview and approach

The Model Context Protocol (MCP) is Anthropic's open standard that acts as "USB-C for AI applications," enabling seamless integration between AI models and external services. **Your goal is ambitious but achievable**: build a multi-API integration tool in a single afternoon using proven patterns and rapid development techniques.

**Key success factors**: leverage existing MCP templates, use the domain-split development pattern, focus on authentication-first setup, and prioritize working functionality over perfect architecture in version 0.1.

## Team structure and parallel development strategy

### Recommended task division: API boundary split

**Developer A ("Integration Specialist")**:

- External API service implementations (`src/services/`)
- Authentication and credential management
- Data transformation utilities
- Rate limiting and error handling
- API client configurations

**Developer B ("Tools Engineer")**:

- MCP tool definitions (`src/tools/`)
- Business logic controllers (`src/controllers/`)
- Response formatting and validation
- MCP resource definitions
- User interface logic

This division creates **natural dependencies** where Developer B depends on Developer A's service interfaces, encouraging clean API design and enabling effective parallel work.

### Git workflow: simplified feature branch

```
bash
```

```
# Initial setup (both developers)
git clone <project-repo>
git checkout -b develop
git push -u origin develop

# Daily workflow
git checkout develop && git pull origin develop
git checkout -b feature/api-service-name    # Dev A
git checkout -b feature/tool-name           # Dev B
# Work → commit → push → pull request → merge to develop daily
```

## Rapid setup guide (first 15 minutes)

### Step 1: Choose your MCP starter template

**Recommended for beginners**: `cyanheads/fastmcp-starter`

```
bash

# Clone the fastest-to-prototype template
git clone https://github.com/cyanheads/fastmcp-starter.git dev-assistant-mcp
cd dev-assistant-mcp
npm install
```

**Alternative options**:

- `alexanderop/mcp-server-starter-ts` (minimal dependencies, auto-loading)
- `aashari/boilerplate-mcp-server` (production-ready with testing)

### Step 2: Environment setup automation

Create `setup.sh`:

```
bash
```

```
#!/bin/bash
# One-command setup script

npm install
cp .env.example .env
mkdir -p logs temp

# Install additional dependencies for your APIs
npm install googleapis @notionhq/client octokit zod dotenv

# Build project
npm run build

echo "✅ Setup complete! Edit .env file with your API credentials"
echo "🔧 Test with: npm run inspect"
```

### Step 3: Essential package.json configuration

```
json

{
  "type": "module",
  "scripts": {
    "dev": "tsx watch --clear-screen=false src/index.ts",
    "build": "tsc && chmod +x dist/index.js",
    "test": "vitest",
    "inspect": "npm run build && npx @modelcontextprotocol/inspector dist/index.js",
    "setup": "./setup.sh"
  },
  "devDependencies": {
    "tsx": "^4.0.0",
    "typescript": "^5.0.0",
    "vitest": "^1.0.0",
    "@types/node": "^20.0.0"
  }
}
```

## API integration implementation guide

### Google Calendar API setup (Developer A - Priority 1)

**Quick authentication setup**: Use service account for fastest POC

```
bash
```

```
# 1. Go to Google Cloud Console → Create project
# 2. Enable Calendar API
# 3. Create service account → Download JSON key
# 4. Share your calendar with service account email
```

**TypeScript implementation**:

```typescript
import { google } from 'googleapis';

export class GoogleCalendarService {
  private calendar;

  constructor(serviceAccountPath: string) {
    const auth = new google.auth.GoogleAuth({
      keyFilename: serviceAccountPath,
      scopes: ['https://www.googleapis.com/auth/calendar']
    });
    this.calendar = google.calendar({ version: 'v3', auth });
  }

  async getEvents(maxResults = 10) {
    const response = await this.calendar.events.list({
      calendarId: 'primary',
      timeMin: new Date().toISOString(),
      maxResults,
      singleEvents: true,
      orderBy: 'startTime'
    });
    return response.data.items;
  }

  async createEvent(eventData) {
    const response = await this.calendar.events.insert({
      calendarId: 'primary',
      requestBody: eventData
    });
    return response.data;
  }
}
```

## Notion API integration (Developer A - Priority 2)

**Setup**: Create integration at notion.com/my-integrations

typescript

```typescript
import { Client } from '@notionhq/client';

export class NotionService {
  private notion;

  constructor(token: string) {
    this.notion = new Client({ auth: token });
  }

  async queryDatabase(databaseId: string, filter = {}) {
    const response = await this.notion.databases.query({
      database_id: databaseId,
      ...(filter && { filter })
    });
    return response.results;
  }

  async createPage(databaseId: string, properties: any) {
    const response = await this.notion.pages.create({
      parent: { database_id: databaseId },
      properties
    });
    return response;
  }

  async readPage(pageId: string) {
    const response = await this.notion.blocks.children.list({
      block_id: pageId
    });
    return response.results;
  }
}
```

### GitHub API integration (Developer A - Priority 3)

**Setup**: Create personal access token at github.com/settings/tokens

typescript

```typescript
import { Octokit } from 'octokit';

export class GitHubService {
  private octokit;

  constructor(token: string) {
    this.octokit = new Octokit({ auth: token });
  }

  async getRepository(owner: string, repo: string) {
    const { data } = await this.octokit.rest.repos.get({ owner, repo });
    return data;
  }

  async getFileContent(owner: string, repo: string, path: string) {
    const { data } = await this.octokit.rest.repos.getContent({
      owner, repo, path
    });

    if ('content' in data && data.encoding === 'base64') {
      return Buffer.from(data.content, 'base64').toString('utf-8');
    }
    return null;
  }

  async listBranches(owner: string, repo: string) {
    const { data } = await this.octokit.rest.repos.listBranches({
      owner, repo
    });
    return data;
  }
}
```

## MCP server implementation (Developer B)

### Core server structure using FastMCP

typescript

```typescript
import { FastMCP } from 'fastmcp';
import { z } from 'zod';
import { GoogleCalendarService } from './services/google-calendar.js';
import { NotionService } from './services/notion.js';
import { GitHubService } from './services/github.js';

const server = new FastMCP({
  name: "Developer Assistant MCP",
  version: "0.1.0"
});

// Initialize services (Developer A's work)
const calendar = new GoogleCalendarService(process.env.GOOGLE_SERVICE_ACCOUNT_KEY!);
const notion = new NotionService(process.env.NOTION_TOKEN!);
const github = new GitHubService(process.env.GITHUB_TOKEN!);

// Tool definitions (Developer B's work)
server.addTool({
  name: "get_calendar_events",
  description: "Get upcoming calendar events",
  parameters: z.object({
    maxResults: z.number().default(10)
  }),
  execute: async ({ maxResults }) => {
    const events = await calendar.getEvents(maxResults);
    return JSON.stringify(events, null, 2);
  }
});

server.addTool({
  name: "query_notion_pages",
  description: "Query pages from a Notion database",
  parameters: z.object({
    databaseId: z.string()
  }),
  execute: async ({ databaseId }) => {
    const pages = await notion.queryDatabase(databaseId);
    return JSON.stringify(pages, null, 2);
  }
});

server.addTool({
  name: "get_github_repo_info",
  description: "Get GitHub repository information",
  parameters: z.object({
    owner: z.string(),
```

```
    repo: z.string()
  }),
  execute: async ({ owner, repo }) => {
    const repoInfo = await github.getRepository(owner, repo);
    return JSON.stringify({
      name: repoInfo.name,
      description: repoInfo.description,
      stars: repoInfo.stargazers_count,
      language: repoInfo.language
    }, null, 2);
  }
});

// Start server
server.start({ transportType: "stdio" });
```

## Version milestone roadmap

### Version 0.1 POC (Target: 4 hours)

**Success criteria**: Evidence of connection to all three services, running locally, terminal interface

**Hour 1**: Setup and authentication

- Clone template, run setup script
- Configure environment variables for all three APIs
- Verify API connections with simple test scripts

**Hour 2-3**: Parallel development

- Dev A: Implement service layer with basic methods
- Dev B: Create MCP tools using Dev A's interfaces
- Both: Write minimal tests for core functionality

**Hour 4**: Integration and testing

- Merge branches, resolve conflicts
- Test with MCP Inspector: `npm run inspect`
- Configure Claude Desktop, verify working connection

**Deliverable**: Working MCP server with 3 basic tools (one per API)

### Version 0.2 Alpha (Target: +2 days)

**Success criteria**: Read calendar events/tasks, read Notion pages, read GitHub main branch, local/MCP server deployment

**Enhanced functionality**:

- Multiple calendar support and filtering
- Notion page content reading with rich formatting
- GitHub repository browsing and file content access
- Error handling and input validation
- Basic logging and debugging

**Testing approach**:

- Unit tests for all service methods
- Integration tests with real APIs (rate-limited)
- End-to-end testing with multiple MCP clients

### Version 0.5 MVP (Target: +1 week)

**Success criteria**: Multiple calendars, Notion read/write, all GitHub branches, proper deployment

**Advanced features**:

- Calendar event creation and modification
- Notion database writes and page updates
- Multi-branch GitHub operations
- Resource definitions for rich context
- HTTP deployment option
- Performance optimization and caching

## Common pitfalls and solutions

### Critical MCP protocol issues

❌ **Never use `console.log()` in stdio servers** - it breaks JSON-RPC communication ✅ **Use `console.error()` or dedicated logging to stderr**

❌ **Hardcoded API credentials** ✅ **Environment variable validation with clear error messages**

### Template-specific gotchas

- **MatthewDailey/mcp-starter**: Uses `.cjs` extension, ensure your imports use `.js` for local files
- **FastMCP**: Requires explicit `transportType: "stdio"` in start() method
- **Official SDK**: Must call `server.connect(transport)` not just instantiate

### Authentication gotchas

- **Google Calendar**: Share calendar with service account email address
- **Notion**: Grant integration access to specific pages/databases
- **GitHub**: Use tokens with appropriate scopes for target repositories

### Build and runtime issues

- **"Cannot find module"**: Check if you're using `.js` extensions in imports (required for ESM)
- **"Unexpected token"**: Ensure `"type": "module"` is in package.json
- **Inspector hangs**: Server might be outputting to stdout - check for stray console.log()
- **Claude can't connect**: Use absolute paths in claude_desktop_config.json

### TypeScript configuration

```json
{
  "compilerOptions": {
    "module": "ES2020",
    "target": "ES2020",
    "moduleResolution": "node",
    "strict": true,
    "esModuleInterop": true
  }
}
```

## Testing and deployment strategies

### Local testing workflow

```bash
```

```
# Test server directly
npm run inspect

# Test with Claude Desktop - add to claude_desktop_config.json:
{
  "mcpServers": {
    "dev-assistant": {
      "command": "node",
      "args": ["/absolute/path/to/dist/index.js"],
      "env": {
        "GOOGLE_SERVICE_ACCOUNT_KEY": "/path/to/key.json",
        "NOTION_TOKEN": "secret_xxx",
        "GITHUB_TOKEN": "ghp_xxx"
      }
    }
  }
}
```

### Unit testing approach

```typescript
import { describe, it, expect } from 'vitest';

describe('Calendar Service', () => {
  it('should fetch events successfully', async () => {
    const calendar = new GoogleCalendarService(testKeyPath);
    const events = await calendar.getEvents(5);
    expect(events).toBeDefined();
    expect(events.length).toBeLessThanOrEqual(5);
  });
});
```

### Deployment options

- **Local development**: Claude Desktop configuration
- **Team sharing**: Docker container with environment variables
- **Production**: NPM package distribution with global installation

## One-afternoon execution plan

### Pre-work (15 minutes)

1. Create Google Cloud project, enable Calendar API
2. Create Notion integration, share target databases

3. Generate GitHub personal access token
4. Clone MCP template and run setup script

### Hour 1: Foundation (Authentication sprint)

- **Both developers**: Set up development environment
- **Dev A**: Implement API authentication for all three services
- **Dev B**: Create basic MCP server structure with placeholder tools
- **Checkpoint**: All API connections working, basic server runs

### Hour 2: Core implementation

- **Dev A**: Build service layer methods (getEvents, queryDatabase, getRepository)
- **Dev B**: Implement MCP tools using service interfaces
- **Integration point**: Dev B tests tools with Dev A's service methods

### Hour 3: Testing and debugging

- **Both**: Test individual components, fix authentication issues
- **Integration**: Merge branches, test complete server
- **MCP testing**: Use Inspector to validate tool functionality

### Hour 4: Deployment and validation

- **Configuration**: Set up Claude Desktop integration
- **End-to-end testing**: Verify all three APIs work through Claude
- **Documentation**: Update README with setup instructions
- **Deliverable**: Working POC demonstrated to stakeholders

### Critical success factors

1. **Authentication first**: Get all API connections working before implementing features
2. **Fail fast**: Use MCP Inspector constantly for immediate feedback
3. **Minimize scope**: Focus on one method per API for POC
4. **Parallel execution**: Stick to domain boundaries to avoid conflicts
5. **Early integration**: Merge and test frequently throughout the day

## Environment configuration template

Create `.env` with these variables:

```env
```

```
# Google Calendar
GOOGLE_SERVICE_ACCOUNT_KEY=/path/to/service-account-key.json

# Notion
NOTION_TOKEN=secret_xxxxxxxxxxxxxxxxx

# GitHub
GITHUB_TOKEN=ghp_xxxxxxxxxxxxxxxxxx

# Development
NODE_ENV=development
LOG_LEVEL=debug
DEBUG=mcp:*
```

## Success metrics and validation

### Version 0.1 POC validation checklist

- [ ] MCP server starts without errors
- [ ] All three API services authenticate successfully
- [ ] Each API has at least one working tool
- [ ] Claude Desktop successfully connects to server
- [ ] Can execute tools through Claude interface
- [ ] Basic error handling prevents server crashes

### Performance expectations

- **Tool response time**: < 5 seconds for simple API calls
- **Server startup**: < 10 seconds
- **Memory usage**: < 100MB for basic operations
- **Error rate**: < 5% for authenticated API calls

## Next steps beyond POC

After achieving version 0.1, your development velocity will significantly increase. The foundation provides:

- **Proven architecture** that scales to additional APIs
- **Development patterns** that new team members can follow
- **Testing framework** for reliable feature additions
- **Deployment pipeline** for seamless updates

Focus areas for version 0.2 include enhanced error handling, richer data transformations, and expanded API coverage. The parallel development model will continue to work effectively as you add complexity.

**Remember**: The goal of version 0.1 is proving feasibility and establishing development momentum. Perfect code can wait - working code creates value and learning opportunities that drive future improvements.