I'll help you create a high-level project plan for an MCP server that wraps the CharityAPI.org API. Let me first research both the CharityAPI and MCP servers to give you the most accurate guidance.Based on my research, I'll provide you with a comprehensive high-level project plan for creating an MCP server that wraps the CharityAPI.org API using Node.js.

**Project Overview**

CharityAPI.org is a simple API for accessing US nonprofit and charity data sourced from the IRS, including charity lookup by EIN, public charity status verification, and searching through ~1.7 million nonprofits. The Model Context Protocol (MCP) is a standardized way for AI applications to connect with external tools and data sources using JSON-RPC 2.0.

**High-Level Project Plan**

**Phase 1: Project Setup & Dependencies**

**1.1 Initialize Node.js Project**

mkdir charity-mcp-server

cd charity-mcp-server

npm init -y

**1.2 Install Core Dependencies**

# MCP SDK and related packages

npm install @modelcontextprotocol/sdk zod

# HTTP client for API calls

npm install axios

# Development dependencies

npm install -D typescript @types/node ts-node

**1.3 TypeScript Configuration** Create a tsconfig.json with ES2022 target, Node16 module resolution, and proper output directory structure

**Phase 2: Core MCP Server Architecture**

**2.1 Server Setup**

* Create McpServer instance with stdio transport for local execution
* Define server capabilities (tools, resources, prompts)
* Implement proper error handling and logging

**2.2 CharityAPI Integration Layer**

* Create API client wrapper for CharityAPI.org endpoints
* Implement authentication/API key management
* Add rate limiting and retry logic
* Handle API response parsing and error handling

**Phase 3: MCP Tools Implementation**

**3.1 Charity Lookup Tool**

* **Purpose**: Look up charity by EIN (Tax ID Number) and return IRS information
* **Input Schema**: EIN/Tax ID (string, required)
* **Output**: Charity details including name, status, location, etc.

**3.2 Public Charity Check Tool**

* **Purpose**: Verify if a nonprofit is a "public charity" with tax-deductible donation status
* **Input Schema**: EIN/Tax ID (string, required)
* **Output**: Boolean public charity status with EIN confirmation

**3.3 Charity Search Tool**

* **Purpose**: Search through ~1.7 million nonprofits in the IRS database
* **Input Schema**: Search query, optional filters (city, state, category)
* **Output**: Array of matching charity records

**Phase 4: Data Processing & Validation**

**4.1 Input Validation**

* Use Zod schemas for all tool inputs
* Validate EIN format (XX-XXXXXXX pattern)
* Sanitize search query strings

**4.2 Response Formatting**

* Standardize output formats across all tools
* Handle API errors gracefully
* Provide meaningful error messages to users

**Phase 5: Configuration & Environment**

**5.1 Environment Configuration**

# .env file

CHARITY\_API\_KEY=your\_api\_key\_here

CHARITY\_API\_BASE\_URL=https://api.charityapi.org

**5.2 Configuration Management**

* API endpoint URLs
* Rate limiting settings
* Timeout configurations
* Debug/logging levels

**Phase 6: Testing & Validation**

**6.1 Unit Tests**

* Test each tool function independently
* Mock CharityAPI responses
* Validate input/output schemas

**6.2 Integration Testing**

* Test with MCP Inspector tool for protocol compliance
* Verify Claude Desktop integration
* End-to-end workflow testing

**Phase 7: Documentation & Deployment**

**7.1 Documentation**

* API usage examples
* MCP server installation instructions
* Configuration guide
* Troubleshooting section

**7.2 Deployment Options**

* **Local**: Stdio transport for Claude Desktop integration
* **Remote**: HTTP transport for cloud deployment on platforms like Cloudflare Workers

**Detailed Implementation Structure**

charity-mcp-server/

├── src/

│ ├── index.ts # Main server entry point

│ ├── tools/ # MCP tool implementations

│ │ ├── charity-lookup.ts

│ │ ├── public-charity-check.ts

│ │ └── charity-search.ts

│ ├── services/ # API integration layer

│ │ ├── charity-api.ts # CharityAPI client

│ │ └── rate-limiter.ts # Rate limiting logic

│ ├── schemas/ # Zod validation schemas

│ │ └── charity-schemas.ts

│ └── utils/ # Helper functions

│ ├── error-handler.ts

│ └── logger.ts

├── tests/ # Test files

├── build/ # Compiled TypeScript

├── .env # Environment variables

├── package.json

├── tsconfig.json

└── README.md

**Key Implementation Considerations**

**1. API Rate Limiting**

* CharityAPI.org responses are typically <80ms, so implement reasonable rate limiting
* Consider caching frequently requested charity data

**2. Error Handling**

* Handle API timeouts, network errors, and invalid EINs gracefully
* Provide clear error messages to help users understand issues

**3. Security**

* Securely manage API keys
* Validate all inputs to prevent injection attacks
* Implement proper CORS if using HTTP transport

**4. Extensibility**

* Design modular architecture for easy addition of new tools
* Support for additional charity data sources if needed

This plan provides a solid foundation for building a robust MCP server that integrates CharityAPI.org with AI applications like Claude Desktop, enabling seamless charity data lookup and verification capabilities.

**Unit Testing**

npm test # Run all tests

npm run test:watch # Watch mode

npm run test:coverage # With coverage report

npm test -- --testPathPattern=schemas # Run specific tests