Generative AI (/tags/generative-ai) Claude Code (/tags/claude-code) Anthropic (/tags/anthropic) AI-assisted Coding (/tags/ai-assisted-coding)

Building an Event Management System with Claude Code: Part 6 - Architecture and Design of a Custom Import MCP Server

Posted by nielsb on Wednesday, January 14, 2026

Welcome back to the **Building an Event Management System with Claude Code** series! If you've been following along, you know we've made significant progress: we've installed Claude Code, integrated it with our IDE, designed an AI-native architecture, set up our cloud database, and created a complete schema with natural language query capabilities.

- **To see all posts in the series, go to: Building an Event Management System with Claude Code (/contact-event-claude-code).**

In Part 5, we accomplished something remarkable, we designed and populated our entire database schema conversationally, and verified we could query it naturally through Claude Desktop. We have contacts, events, tickets, speaker sessions, and ratings all properly structured and queryable with natural language. In Part 5.5, we refined our schema based on actual CSV exports from Quicket and Sessionize, a crucial lesson in letting real data drive design.

But there's a problem: our database only has test data.

The Pain Point

Here's my reality before building this system: After each Data & AI Community Day Durban event, I'd spend 2-3 hours manually wrangling data:

- Export registration CSV from Quicket (ticketing platform)

- Export check-in CSV from Quicket
- Export speaker data from Sessionize
- Export feedback responses from Microsoft Forms
- Export walk-in check-ins CSV Microsoft Forms
- Open Excel, copy-paste, deduplicate, format, manually match records
- Finally import into Brevo for email communications

Manual, error-prone, soul-crushing work. The kind of task that makes you think “there has to be a better way.”

Today, we’re starting on building that better way: a conversational import workflow where I can simply say:

```
Me: "Import the Quicket registrations from the March 2025 event"

Claude Desktop: "Found quicket-march-2025.csv. Analyzing...
147 registrations found. 3 appear to be duplicates of existing contacts.
Should I update the existing records or skip them?"

Me: "Update existing records"

Claude Desktop: "Imported successfully:
• 144 new contacts created
• 3 existing contacts updated
• 147 tickets recorded for March 2025 event
• 12 group purchases identified (shared order numbers)"
```

Code Snippet 1: *Conversational Import Workflow*

No Excel. No manual matching. Just conversation.

Quick Recap: Where We Are

Let’s recap our journey so far:

Part 1-3: Foundation

- Installed Claude Code on macOS/Windows
- Integrated with VSCode/IDEs
- Architected an AI-native system (conversational interface, MCP servers)

Part 4: Infrastructure

- Set up PostgreSQL database on Neon (cloud-hosted, free tier)
- Installed Postgres MCP Pro (bridge between Claude and database)
- Verified natural language database queries work

Part 5: Schema Design

- Designed initial database schema with Claude Code
- Created tables for contacts, events, tickets, sessions, and ratings
- Generated test data and tested natural language queries

Part 5.5: Schema Refinement ⭐

- **This was crucial.** I realised I'd designed the schema from my mental model instead of from actual data
- Fed real CSV files from Quicket and Sessionize to Claude Code
- Discovered 13 missing fields (order numbers, ticket types, speaker IDs, tracks, etc.)
- Generated and ran a migration to add the missing fields
- **Key lesson learned:** Show AI your real-world constraints first, then design together

Now our schema perfectly matches our import sources. This means the MCP server we're designing today will have clean, straightforward import logic with no workarounds or hacks needed.

What We'll Accomplish Today

By the end of this post, you'll have:

- ✅ Deep understanding of import challenges from real CSV formats
- ✅ Critical MCP architecture insight (server isolation)
- ✅ Complete tool specification (7 tools, rationale, build order)
- ✅ Claude Code hygiene practices (CLAUDE.md management, compacting)
- ✅ Design methodology using AI-assisted architecture discussions
- ✅ Ready-to-implement blueprint for our Import MCP server

Time investment: ~2-3 hours of AI-assisted development
Time saved per event: ~2-3 hours of manual work (break-even after first import!)

The Paradigm Shift: Dual Interface

Before we dive in, let me clarify how this AI-native system works, because it's different from traditional development:

Traditional Approach:

```
Developer writes code → Deploys web app → Users click through forms
```

Code Snippet 2: *Traditional Development Workflow*

AI-Native Approach (what we're building):

```
Developer builds MCP server ← Claude Code assists
     ↓
Claude Desktop uses MCP server ← User has conversation
```

Code Snippet 3: *AI-Native Development Workflow*

Two distinct interfaces:

1. **Claude Code** (Development Interface)
    - Where we build the MCP server
    - Where we write the import tools
    - Where we test and debug
2. **Claude Desktop** (User Interface)
    - Where we actually import data
    - Where we ask "import this file"
    - Where we get conversational feedback

The MCP server is the bridge between conversational requests and database operations.

Series Progress

Here's where we are in the journey:

- ✅ Part 1: Installation & Setup - Claude Code on macOS/Windows (/post/2025-07-29-building-an-event-management-system-with-claude-code-claude-code-installation-and-initialisation/)

- ✅ Part 2: IDE Integration - VSCode workflows and advanced features (/post/2025-08-13-building-an-event-management-system-with-claude-code-part-2---ide-integration-and-advanced-features/)
- ✅ Part 3: Architecture Planning - AI-native design and MCP strategy (/post/2025-12-28-building-an-event-management-system-with-claude-code-part-3---architecting-an-ai-native-system/)
- ✅ Part 4: Database Infrastructure - PostgreSQL on Neon, MCP Pro setup (/post/2026-01-01-building-an-event-management-system-with-claude-code-part-4---database-setup-and-first-conversations/)
- ✅ Part 5: Schema Design - Tables, natural language queries, test data (/post/2026-01-04-building-an-event-management-system-with-claude-code-part-5---database-schema-and-natural-language-queries/)
- ✅ Part 5.5: Schema Refinement - Iterating based on real CSV files (/post/2026-01-12-building-an-event-management-system-with-claude-code-part-55---schema-refinement-when-real-data-reveals-the-truth/)
- 🚀 **Part 6: Architect & Design of a Custom Import MCP Server (this post)**
- 🗓️ Part 7: Implementing the Custom Import MCP Server
- 🗓️ Part 8: Email Integration - Brevo MCP and communication workflows
- 🗓️ Part 9: Production Usage - Real events, analytics, insights

With our refined schema in place and all infrastructure ready, we're designing the import pipeline that brings real-world data into our AI-native system.

> ***NOTE:*** *I cannot guarantee that the subsequent posts will be as per above. Part 5.5 was not initially planned but became necessary based on real data. Future parts may also shift based on learnings.*

Let's get started.

Understanding the Import Challenge

Before we start designing, let me show you what we're dealing with and why it is challenging. These are actual export formats from the platforms we use (anonymised).

Quicket Registration Export (CSV)

A typical Quicket export looks like this:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Order Nur	Ticket Num	Purchase Date	Checked	Comp	EventDate	Link Camp	Ticket Num	Cellphone	Email	First name	Surname	Profession:
2	16605830	42973236	2025-04-21 19:	No	-	2025-05-24	Email	42973236	+27816789012	priya.naidoo@gmail-co-za.co.za	Priya	Naidoo	Employed
3	16605898	42973373	2025-04-21 19:	No	-	2025-05-24	Email	42973373	+27814567890	liesl.fourie@email.co.za	Liesl	Fourie	Student
4	16606679	42975114	2025-04-21 21:	No	-	2025-05-24	LinkedIn	42975114	+27823456789	hendrik.v@myrain.co.za	Hendrik	Viljoen	Employed
5	16606679	42975115	2025-04-21 21:	No	-	2025-05-24	LinkedIn	42975115	+27823456789	hendrik.v@myrain.co.za	Hendrik	Viljoen	Employed
6	16606391	42974402	2025-04-21 20:	No	-	2025-05-24	Email	42974402	+27838901234	rebecca.stone@ibm-1.com	Rebecca	Stone	Student
7	16605545	42972667	2025-04-21 18:	No	-	2025-05-24	Email	42972667	+27834567890	kobus.vh@up.ac.za	Kobus	van Heerden	Employed
8	16605545	42972668	2025-04-21 22:	No	-	2025-05-24	Email	42972668	+27834567890	kobus.vh@up.ac.za	Sarah	van Heerden	Student
9	16605162	42971945	2025-04-21 17:	No	-	2025-05-24	Email	42971945	+27810123456	chloe.williams@email.co.za	Chloe	Williams	Student
10	16606545	42974759	2025-04-21 21:	No	-	2025-05-24	LinkedIn	42974759	+27834567890	sarah@myrain.co.za	Sarah	Phillips	Employed

Figure 1: *Sample Quicket Registration Export CSV*

In *Figure 1*, you see some of the key fields the file contains (there are 30+ columns).

Challenges:

- One physical person can “order” multiple tickets (e.g. family/friends); you see examples of that outlined in yellow and blue.
    - If we don’t have this person as a contact yet, we need to create a new contact record.
    - However, in the case of duplicates where the name is different (blue), but the same email address. In this case, we need to add a second contact (unless one already exists). This could be a spouse or a friend using the same email address.
    - Regardless of whether it’s one or multiple contacts, we need to create a ticket record for each ticket purchased, linked to the correct contact.
- Multiple tickets can share the same `Order Number` (group purchases)
- `Purchaser Email` may differ from the attendee’s `Email`
- `Ticket Type` varies: Free, Complimentary, etc

Quicket Check-in Export (CSV)

The Quicket check-in export looks like this:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Order Nur	Ticket Num	Purchase Date	Checked	Comp	EventDate	Link Camp	Ticket Num	Cellphone	Email	First name	Surname	Profession:
2	16605830	42973236	2025-04-21 19:	Yes	-	2025-05-24	Email	42973236	+27816789012	priya.naidoo@gmail-co-za.co.za	Priya	Naidoo	Employed
3	16605898	42973373	2025-04-21 19:	No	-	2025-05-24	Email	42973373	+27814567890	liesl.fourie@email.co.za	Liesl	Fourie	Student
4	16606679	42975114	2025-04-21 21:	Yes	-	2025-05-24	LinkedIn	42975114	+27823456789	hendrik.v@myrain.co.za	Hendrik	Viljoen	Employed
5	16606679	42975115	2025-04-21 21:	Yes	-	2025-05-24	LinkedIn	42975115	+27823456789	hendrik.v@myrain.co.za	Hendrik	Viljoen	Employed
6	16606391	42974402	2025-04-21 20:	No	-	2025-05-24	Email	42974402	+27838901234	rebecca.stone@ibm-1.com	Rebecca	Stone	Student
7	16605545	42972667	2025-04-21 18:	Yes	-	2025-05-24	Email	42972667	+27834567890	kobus.vh@up.ac.za	Kobus	van Heerden	Employed
8	16605545	42972668	2025-04-21 22:	Yes	-	2025-05-24	Email	42972668	+27834567890	kobus.vh@up.ac.za	Sarah	van Heerden	Student
9	16605162	42971945	2025-04-21 17:	No	-	2025-05-24	Email	42971945	+27810123456	chloe.williams@email.co.za	Chloe	Williams	Student
10	16606545	42974759	2025-04-21 21:	Yes	-	2025-05-24	LinkedIn	42974759	+27834567890	sarah@myrain.co.za	Sarah	Phillips	Employed

Figure 2: *Sample Quicket Check-in Export CSV*

You can see in *Figure 2* that this file is identical to the registration file, but it shows a `Yes` in the `Checked In` column for those who attended.

The check-in is pretty straightforward: we need to match the ticket number to the ticket record, and update the `checked_in` field to true. The one thing to be aware of is that some tickets in the check-in export might not yet be in our database (an edge case: late registration). i.e., we need to create a new contact and ticket record if the ticket number is not found.

Microsoft Forms Walk-Ins Export (CSV)

There are always walk-ins at the events (i.e. not registered on Quicket). For these, we use a simple Microsoft Forms survey to capture their details. The export looks like this:

	A	B	C	D	E
1	ID	First Name	Last Name	Email Address	Cell Phone Number
2	1	Nkosinathi	Dube	nkosinathi.dube@multi.co.za	+27829012345
3	2	Mpho	Sithole	mpho.sithole@email.co.za	+27841234567
4	3	Rachel	Wood	rachel.wood@wood.co.za	+27818901234
5	4	Sipho	Ndlovu	sipho@email.co.za	+27845678901

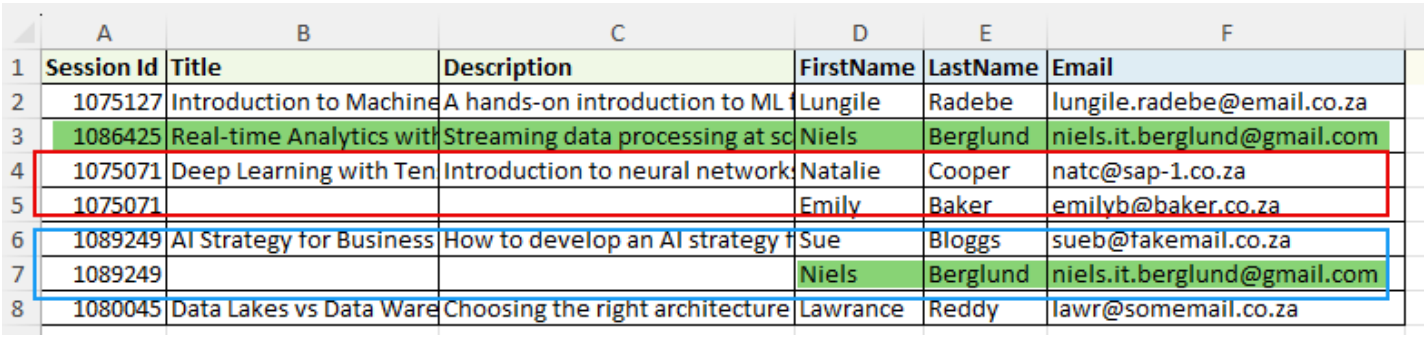
Figure 3: *Sample Microsoft Forms Walk-Ins Export CSV*

Challenges:

- Since these attendees have not registered on Quicket (at least for this event), we do not have a ticket number for them (fortunately, the `tickets` table defaults to `-1`).
    - We need to create a new contact record for each walk-in, if they are not already in our contacts.
    - We also need to create a new ticket record, linked to the contact, with the `checked_in` field set to true.
    - We need to link the ticket to the correct event.

Sessionize Speaker Export (CSV)

The Sessionize speaker export looks like this:



	A	B	C	D	E	F
1	**Session Id**	**Title**	**Description**	**FirstName**	**LastName**	**Email**
2	1075127	Introduction to Machine	A hands-on introduction to ML f	Lungile	Radebe	lungile.radebe@email.co.za
3	1086425	Real-time Analytics with	Streaming data processing at sc	Niels	Berglund	niels.it.berglund@gmail.com
4	1075071	Deep Learning with Ten	Introduction to neural network	Natalie	Cooper	natc@sap-1.co.za
5	1075071			Emily	Baker	emilyb@baker.co.za
6	1089249	AI Strategy for Business	How to develop an AI strategy f	Sue	Bloggs	sueb@fakemail.co.za
7	1089249			Niels	Berglund	niels.it.berglund@gmail.com
8	1080045	Data Lakes vs Data Ware	Choosing the right architecture	Lawrance	Reddy	lawr@somemail.co.za

Figure 4: *Sample Sessionize Speaker Export CSV*

Challenges:

- Each speaker may have multiple sessions (see green highlight).
  - Need to deduplicate speakers while creating multiple session records
- Some speakers may already exist in our contacts (need to match by email)
- Sessions may have multiple speakers, which you can see is outlined in red and blue. We need to link the session to multiple contacts.
  - This becomes a challenge for speaker ratings.

Microsoft Forms Speaker Ratings Export (CSV)

The speaker evaluations come from an “in-house” Microsoft Forms survey and looks like so:

	A	B	C	D	E	F	G
1	ID	Select the session you attended	Session Rating	Speaker Rating	Comments on Speaker	Event Rating	Comments on Event
2	1	03. Introduction to Machine Learning with Python - Lungile Radebe	6	3	Very interesting topic and discussion but the slides weren't really visible and the speaker was monotonous	4	
3	2	01. Real-time Analytics with Apache Kafka - Niels Berglund	7	7	The speaker was very clear and very informative. I would definitely	7	Nothing so far . All is well.
4	3	02. Data Lakes vs Data Warehouses - Lawrance Reddy	7	7		7	
5	4	02. Data Lakes vs Data Warehouses - Lawrance Reddy	7	7		7	
6	5	02. Data Lakes vs Data Warehouses - Lawrance Reddy	7	7		7	Maybe include a demo
7	6	03. Introduction to Machine Learning with Python Lungile Radebe	4	5		5	
8	7	03. Introduction to Machine Learning with Python - Lungile Radebe	7	7	none	7	none
9	8	03. Introduction to Machine Learning with Python Lungile Radebe	7	7		7	
10	9	01. Real-time Analytics with Apache Kafka - Niels Berglund	7	7	Absolutely well spoken and learnt so much.	7	Nothing
11	10	04. AI Strategy for Business Leaders - Sue Bloggs, Niels Berglund	6	6		5	
12	11	07. Deep Learning with TensorFlow - Natalie Cooper, Emily Baker	7	6		7	
13	12	04. AI Strategy for Business Leaders - Sue Bloggs, Niels Berglund	6	7	The speaker was awesome, very interactive.	7	
14	13	07. Deep Learning with TensorFlow - Natalie Cooper, Emily Baker	6	6		5	
15	14	07. Deep Learning with TensorFlow - Natalie Cooper, Emily Baker	5	4	too rushed...	5	session should have been 45 minutes rather than 30..
16	15	04. AI That Makes Money: Building a Lead Engine for Sales with Azure OpenAI - Fanie Ndlovu	6	6	N/A	6	Speaker was very clear and concise
17	16	07. Deep Learning with TensorFlow - Natalie Cooper, Emily Baker	7	7	Very interactive session. Really enjoyed the pair.	6	

Figure 5: *Sample Microsoft Forms Speaker Ratings Export CSV*

Challenges:

- There is one set of ratings for a session, even if there are multiple speakers. You see that outlined in red in *Figure 5*. We need to link the rating to all speakers for that session.
- The session titles in the feedback form may not exactly match the session titles in Sessionize (typos, abbreviations). We may need to do fuzzy matching to link the rating to the correct session.

Why This is Complex

Traditional import problems:

1. **Format inconsistencies** - CSV encodings, date formats, phone number formats
2. **Duplicate detection** - Same person registered multiple times
3. **Data validation** - Invalid emails, missing required fields
4. **Relationship matching** - Linking attendees to events, sessions to speakers

5. **Error handling** - Partial failures, rollback strategies

Our additional complexity: 6. **Group purchases** - Multiple tickets per order, different attendees from the purchaser 7. **Speaker/attendee duality** - Same person might attend one year, speak the next 8. **Fuzzy matching** - Feedback forms don't have exact session IDs 9. **Cross-year tracking** - Speaker retention using Sessionize IDs

This is exactly the kind of domain-specific logic that makes a custom MCP server valuable.

What we need:

A custom MCP server that bridges the gap and uses existing tools where possible, adds custom logic for domain-specific needs. All three sources (Quicket, Sessionize, Microsoft Forms) export CSV/Excel formats, but each has unique column structures, data relationships, quality issues, and business rules.

The Architecture: Self-Contained Import MCP Server

Let me clarify the architecture so you can understand how MCP servers work.

How MCP Servers Actually Work

Critical architectural fact: MCP servers **cannot call each other**. Each MCP server is an isolated process that communicates only with the LLM host (Claude Desktop/Claude Code).

> ***NOTE:*** *In previous parts of this series, we may have implied that MCP servers can call each other. This is incorrect. MCP servers are isolated and do not have direct communication channels.*

The LLM is the orchestrator, not the MCP servers:

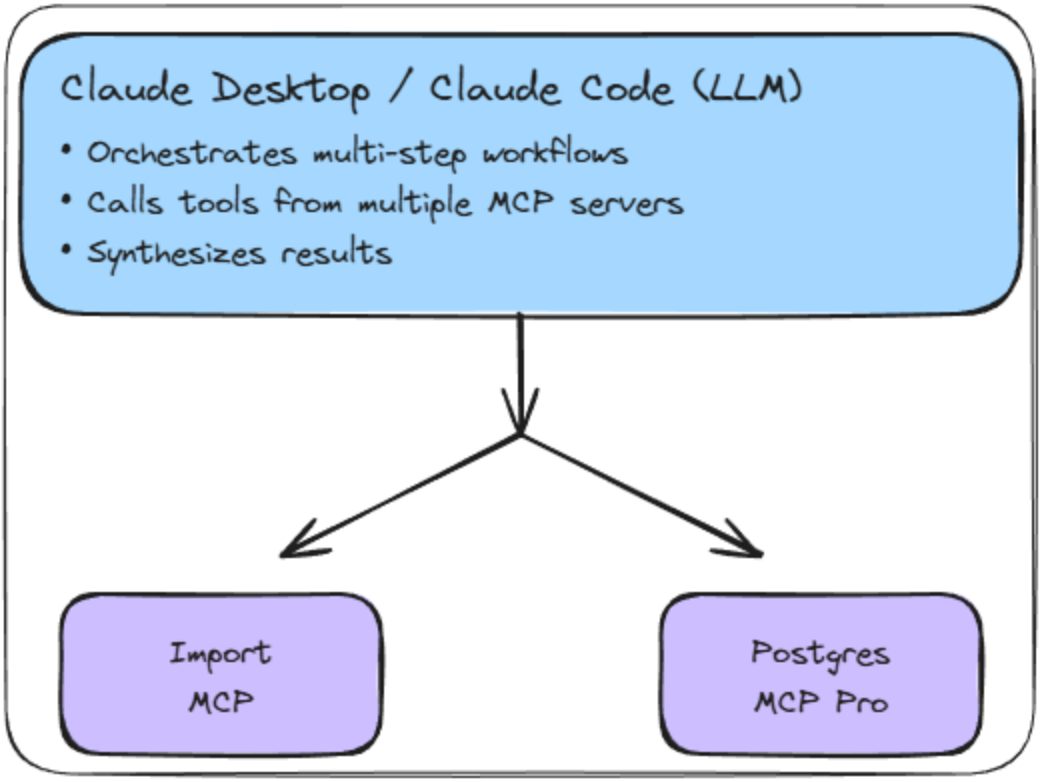


Figure 6: *Claude as Orchestrator*

We see in *Figure 6* that the user interacts with Claude Desktop/Code, which, upon request, calls the MCP servers as needed. Each MCP server operates independently and cannot invoke tools from another MCP server.

Our Design Decision: Self-Contained Server

Given this constraint, we have two architectural options:

Option A: Multi-Step Orchestration:

```
User: "Import Quicket registrations"
│
├─ Claude calls: csv-mcp.analyze_file()
├─ Claude calls: import-mcp.validate_format()
├─ Claude calls: import-mcp.transform_data()
└─ Claude calls: postgres-mcp.execute_sql()
```

Code Snippet 4: *Option A: Multi-Step Orchestration*

Problems:

- ❌ Multiple tool calls (slower)
- ❌ LLM manages state between calls

- ❌ Higher token usage
- ❌ More complex conversation flow

This option is doable but cumbersome. The LLM has to manage multiple steps, keep track of state, and orchestrate the entire import process. This increases complexity and potential for errors.

Option B: Self-Contained Import MCP:

```
User: "Import Quicket registrations"
│
└─ Claude calls: import-mcp.import_quicket_registrations()
   │
   └─ Returns: Complete results in one call
```

Code Snippet 5: *Option B: Self-Contained Import MCP*

What we see in *Code Snippet 5* looks much cleaner.

Benefits:

- ✅ Single tool call per import (fast, simple)
- ✅ Atomic operations (import succeeds or fails as a unit)
- ✅ Simpler for LLM to use
- ✅ All import logic encapsulated

We chose Option B. Our Import MCP server will be self-contained, handling all aspects of the import process internally. This keeps the user experience smooth and the architecture clean.

What “Leveraging Existing Tools” Really Means

When we previously said we’d “leverage existing MCP servers,” what we actually mean is:

We use the same battle-tested Python libraries:

- **pandas** - CSV parsing and data manipulation (same library csv-mcp-server uses)
- **psycopg2** - PostgreSQL database operations (same library Postgres MCP Pro uses)
- **FastMCP** - MCP server framework (same framework many MCP servers use)

We don’t call other MCP servers; we use the same libraries they use.

The 80/20 Principle Still Applies

80% of the work is done by proven libraries:

- pandas handles CSV parsing, data validation, and transformation
- psycopg2 handles database connections, transactions, and SQL execution
- FastMCP handles MCP protocol, tool registration, and parameter validation

20% is our custom code:

- Quicket-specific format validation
- Sessionize-specific data transformation
- Event management business logic
- Duplicate detection strategy
- Error messages tailored to our domain

Architecture Diagram

Taking the above into account, here's the overall architecture:

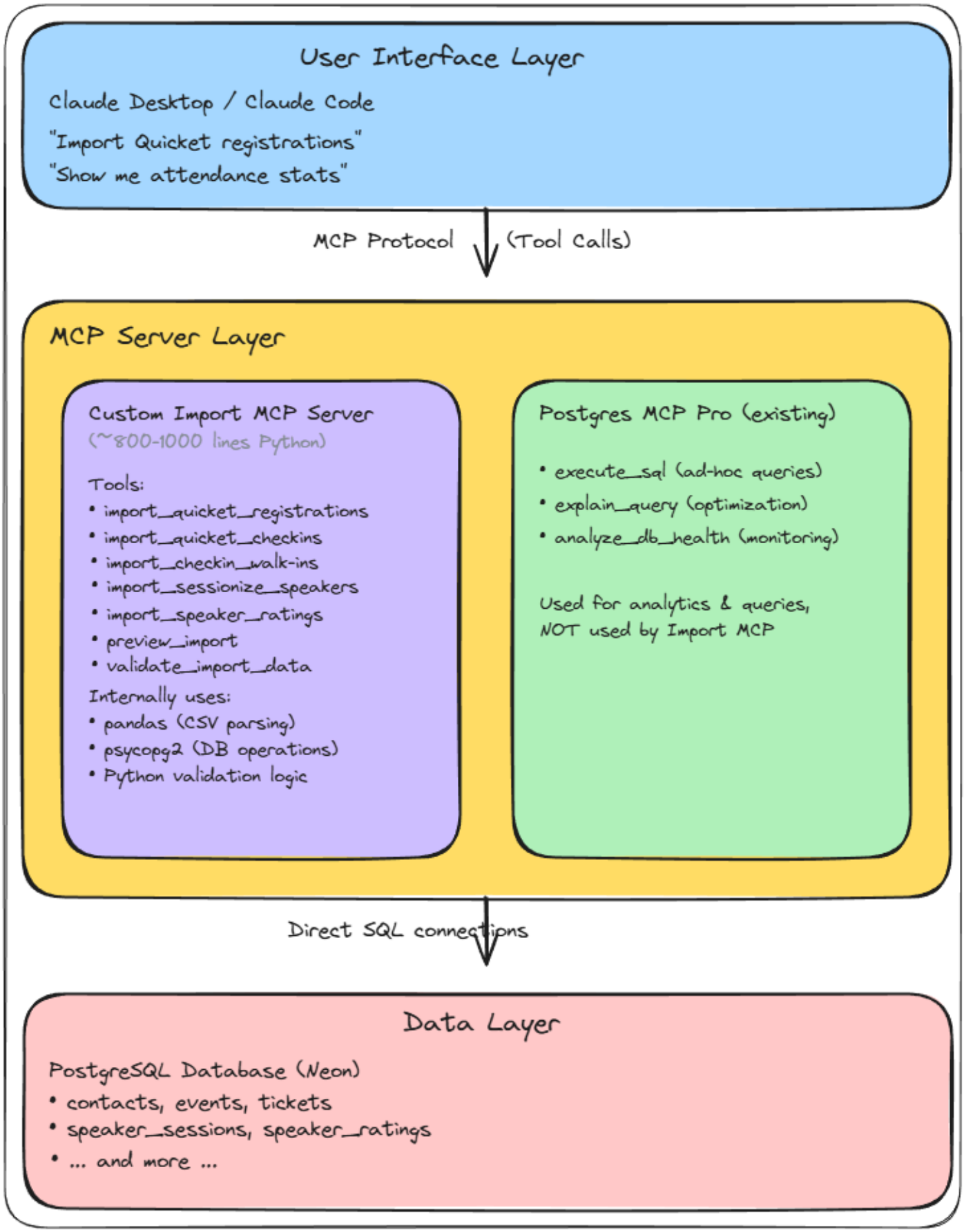


Figure 7: *Overall Architecture with Custom Import MCP Server*

In *Figure 7*, we see the following **key points:**

- Import MCP and Postgres MCP Pro are **separate, independent** servers
- Import MCP connects **directly** to PostgreSQL (doesn't use Postgres MCP)
- Postgres MCP Pro is used for **ad-hoc natural language queries**
- Claude Desktop orchestrates which server to use based on user intent

Code Size Reality Check

Given the solution we've discussed, let's set realistic expectations for the code size of our Import MCP server.

- **Initial estimate:** ~500-800 lines
- **Revised estimate:** ~800-1000 lines

Why larger:

- Self-contained logic (not calling other MCP servers)
- More comprehensive error handling
- Domain-specific validation rules
- Transaction management
- Duplicate detection strategies
- Logging and debugging support

But still very manageable because we're using pandas and psycopg2 to do the heavy lifting.

Designing the Import MCP Server with Claude Code

Now that we've laid the groundwork, let's design our Import MCP server step by step using Claude Code.

However, before we start, there's one important thing to address.

Claude Code Hygiene

When I started Claude Code with `claude --continue`, I was greeted with this:

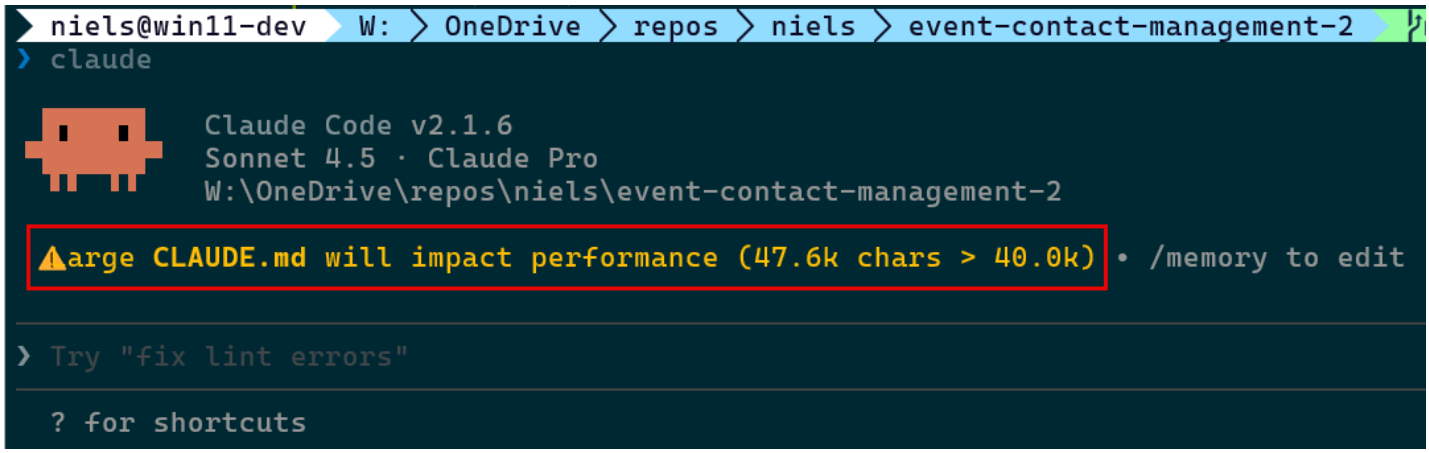


Figure 8: *Claude Code large CLAUDE.md File*

As shown in *Figure 8* (outlined in red), Claude Code complains that `CLAUDE.md` is large and could slow performance. Best practices are to file as small as possible. We should avoid loading everything into Claude Code if we can. So, I did:

```
Large CLAUDE.md will impact performance (47.6k chars > 40.0k). Can you reduce the size of CLAUDE.m
```

Code Snippet 6: *Reducing CLAUDE.md Size*

Claude Code responded:

```
● I'll help you condense the CLAUDE.md file while preserving all critical information. The main ar

  1. Research findings sections (Email/CSV/PostgreSQL) - These are extremely detailed with alterna
  2. Verbose explanations - Many sections have redundant or philosophical content
  3. Conversation examples - Can be shortened while keeping the key patterns

  Let me create a condensed version:

[snipped for brevity]

● I'll create a condensed version that removes verbose research findings, redundant explanations,

[Claude Code works on condensing the file - snipped for brevity]

● Perfect! I've successfully reduced your CLAUDE.md file from 47,600 characters to 15,158 characte

  What I Removed:

  1. Detailed research findings (~12k chars) - Removed extensive comparisons of alternative email
  2. Verbose explanations (~8k chars) - Condensed redundant descriptions in Core Components, Imple
  3. "Potential Challenges & Mitigations" (~2k chars) - Removed entirely as it was speculative.
  4. Philosophical content (~3k chars) - Condensed "Schema Design Lessons Learned" from 4k to 1k c
  5. Lengthy conversation examples (~2k chars) - Reduced from detailed multi-step flows to concise

  What I Preserved (All Critical Info):

  ✅ Architecture diagram (visual reference)
  ✅ 10 core tables with key fields
  ✅ Selected MCP servers with installation commands
  ✅ Implementation status (what's done/pending)
  ✅ Technology stack table
  ✅ Next immediate steps
  ✅ Example workflows (condensed)
  ✅ Key advantages & why LLM/MCP approach wins

  The condensed version is now much more scannable while retaining 100% of the actionable informat
```

Code Snippet 7: *Claude Code's Condensed CLAUDE.md Response*

This was a great exercise in keeping things lean. If your `CLAUDE.md` file grows large, consider asking Claude Code to help you condense it while preserving critical information.

Compacting

Apart from keeping Claude Code lean, we can reduce the size of the current conversation history by summarising it, so that you can continue working without hitting the model's context (memory) limit. We do that by the `/compact` command. Later versions of Claude Code do this automatically when the context limit is approached.

OK, let's get on with it!

Initial Design Discussion

By now, Claude Code should have a solid understanding of our project. Let's start by discussing the design of the Import MCP server. Instead of me telling Claude what to do, let us ask Claude Code to think about what we need:

```
We are now ready to continue with the Import MCP Server. I want youto THINK Hard about what is nee
```

Code Snippet 8: *Design Discussion Prompt*

I gave Claude Code some time to think, and it came back with a detailed breakdown of the components we need, the challenges we might face, and a high-level implementation plan.

```
● I'll think through what's needed for the Import MCP Server systematically.

  Let me first explore the codebase to understand what documentation and sample data already exist

[snipped for brevity]

● Perfect! Based on the comprehensive exploration, I now have a complete understanding of what's n

  Import MCP Server - Complete Requirements Analysis

[claude code doing analysis - snipped for brevity]

  ---
  Next Steps - What Would You Like to Do?

  I see several options:

  Option A: Start Building (Recommended)

  1. Set up FastMCP project structure with pyproject.toml
  2. Implement database utilities first (utils/db.py, utils/parsers.py)
  3. Build tools incrementally (validation → quicket → sessionize → ratings)
  4. Test with real CSV files as we go

  Option B: Deep Dive First

  1. Read the full 32,000-word import analysis document
  2. Review each CSV file in detail
  3. Create detailed field mapping specifications
  4. Then build with maximum precision

  Option C: Prototype One Tool

  1. Build just import_quicket_registrations() as proof-of-concept
  2. Test with real reg.csv file
  3. Validate approach before building remaining 6 tools

  Option D: Review & Refine

  1. You review my analysis above
  2. Point out any gaps or concerns
  3. We refine the design before coding

  Which approach would you prefer? I'm ready to start coding whenever you are, or we can discuss t
```

Code Snippet 9: *Claude Code's Design Analysis Response*

As shown in *Code Snippet 9*, Claude Code provided a detailed breakdown of the components we need, the challenges we might face, and a high-level implementation plan. However, when I reviewed the analysis, I became confused about some of the details, specifically Option A, point 3. So, I asked for clarification:

```
I like Option A. However, in point 3, you mention "Build tools incrementally (validation → quicket
```

Code Snippet 10: *Clarification Prompt*

After thinking a bit, Claude Code responded:

```
● Excellent question! Let me clarify the tool architecture with clear reasoning.

[snipped for brevity]

● Perfect! Based on the 32,000-word import analysis document, here's the exact tool specification:

  7 MCP Tools to Build

  Core Infrastructure (Build First)

  1. validate_import_data

[snipped for brevity]

  2. preview_import

[snipped for brevity]
  ---
  Import Operations (Build by Priority)

[snipped for brevity]
  ---
  Recommended Build Order

  Phase 1: Foundation + High-Value Imports (Build this first)

[snipped for brevity]

  Phase 2: Enhanced Operations (Build after Phase 1 works)

[snipped for brevity]
  ---
  Key Design Decisions

  Why 7 separate tools vs fewer unified tools?

  ✅ Separate because:
  - Different CSV formats (37 columns vs 10 columns vs 25 columns)
  - Different operations (CREATE vs UPDATE vs LINK)
  - Different workflows (pre-event vs post-event)
  - Different complexity (simple walk-ins vs complex multi-speaker sessions)
  - Follows MCP principle: "Atomic tools (one thing well), composable"

  Why validate + preview as separate tools?

  ✅ Separate because:
  - validate_import_data = fast, no DB access (50ms)
  - preview_import = slower, reads DB (500ms+)
  - Composable: Can validate multiple files, then preview one
  - User can validate quickly, then decide which file to import

  ---
```

```
Summary

Total: 7 tools, ~1,250 lines of code

- Infrastructure: 350 lines (validation + preview)
- Phase 1 imports: 450 lines (quicket registrations + ratings)
- Phase 2 imports: 450 lines (check-ins + walk-ins + sessionize)

Incremental build path: Infrastructure → Quicket registrations → Ratings → Check-ins → Walk-ins

Does this clarify the tool architecture? Ready to start building Phase 1 when you are!
```

Code Snippet 11: *Claude Code's Tool Specification Response*

As you see in *Code Snippet 11*, Claude Code provided a clear breakdown of the 7 MCP tools we need to build, along with the rationale for the design decisions. This clarity allowed me to proceed confidently with the implementation.

> ***NOTE:*** *The above design discussion is a simplified version of the actual conversation. In reality, it took multiple back-and-forth exchanges to arrive at this level of clarity. When we were finished, I asked Claude Code to document the entire design discussion in a separate markdown file for future reference. You find it as a* `gist` *at* ***Import MCP Server - Tool Specification*** *(https://gist.github.com/nielsberglund/b26fd60e18abb4a8e7ca0157a7fc69e0).*

When this design discussion was complete, I felt ready to start building the MCP server. In the next post, we'll dive into the actual implementation, step by step.

What We've Accomplished

Today we did the hard thinking that makes implementation straightforward:

Architecture Clarity:

- ✅ Confirmed MCP servers are isolated (can't call each other)
- ✅ Chose self-contained design over multi-step orchestration
- ✅ Defined clear separation: Import MCP (writes) vs Postgres MCP Pro (queries)

Complete Specification:

- ✅ 7 tools defined with clear responsibilities
- ✅ Build order established (validation → high-value imports → enhanced operations)
- ✅ Code size estimated realistically (800-1000 lines)

Design Documentation:

- ✅ Import challenges catalogued with real CSV examples
- ✅ Tool specification documented ([gist link])
- ✅ Architecture diagram created

Claude Code Mastery:

- ✅ Learned CLAUDE.md hygiene (reduced from 47k to 15k chars)
- ✅ Used /compact to manage context limits
- ✅ Leveraged Claude Code for design thinking, not just coding

Why Design Before Implementation Matters

In traditional development, we might have jumped straight into code. But with AI-native development, the design conversation IS the development process. The 30-minute discussion with Claude Code gave us:

- Clarity on MCP constraints (isolation insight)
- Tool boundaries and responsibilities
- Implementation roadmap
- Realistic effort estimates

This upfront thinking means our implementation will be straightforward, not exploratory.

What's Next: Part 7 - Implementation

In the next post, we'll bring this design to life:

Part 7: Building a Custom Import MCP Server - Implementation and Testing

- Set up FastMCP project structure
- Implement the 7 tools (starting with validation, then Quicket registrations)
- Test with real CSV files from actual events

- Configure in Claude Desktop
- Experience conversational imports: “Import the March 2025 registrations”

The design is complete. Now we build.

Sneak Peek: Part 7

Me: “Import the Quicket registrations from March 2025”

Claude Desktop: “Analyzing quicket-march-2025.csv… 147 registrations found, 3 duplicates. Update existing?”

Me: “Update them”

Claude Desktop: “✅ Import complete:

- 144 new contacts created
- 3 contacts updated
- 147 tickets recorded”

No Excel. No copy-paste. Just conversation.

See you in Part 7 where we make this real.

~ Finally

That’s all for this post! We covered a lot of ground today, but it was all about thinking before building - a crucial step that AI-native development makes both easier and more thorough.

What we accomplished:

- Deep understanding of our import challenges (real CSV formats, real complexity)
- Critical architectural insight: MCP servers are isolated, LLM orchestrates
- Complete tool specification: 7 tools with clear responsibilities and build order
- Claude Code hygiene practices that keep development smooth
- A ready-to-implement blueprint

The key insight: MCP servers can't call each other. This fundamentally changed our architecture from "hybrid orchestration" to "self-contained server." Getting this right in design saves hours of debugging later.

Next time: We roll up our sleeves and build. Part 7 will be hands-on: FastMCP setup, implementing the 7 tools, testing with real CSV files, and finally experiencing conversational imports: "Import the March 2025 registrations."

The design is complete. The implementation will be straightforward because we did the hard thinking first.

Your Turn

If you're following along:

1. Review your own import workflows - what CSV/Excel exports do you work with?
2. Think about which MCP servers you'd need (self-contained vs orchestrated)
3. Have the design conversation with Claude Code before writing any code

The architecture discussion alone is worth the time investment.

Have questions or thoughts?

Ping me: niels.it.berglund@gmail.com (mailto:niels.it.berglund@gmail.com)
Follow on LinkedIn: linkedin.com/in/nielsberglund (https://www.linkedin.com/in/nielsberglund/)

Found this helpful? Share it with your network! The AI-native development journey is one we're all taking together, and the architectural insights matter as much as the code.

See you in Part 7 where we build this thing!

← PREVIOUS POST (/POST/2026-01-12-BUILDING-AN-EVENT-MANAGEMENT-SYSTEM-WITH-CLAUDE-CODE-PART-55---SCHEMA-REFINEMENT-WHEN-REAL-DATA-REVEALS-THE-TRUTH/)


(mailto:niels.it.berglund@gmail.com)


(https://twitter.com/nielsberglund)


(https://github.com/nielsberglund)


(https://www.linkedin.com/in/niels-berglund-0122593)


(https://medium.com/@nielsberglund)


(https://stackoverflow.com/users/7656880/niels-berglund)


(https://fosstodon.org/@nielsb)


(https://nielsberglund.com/index.xml)

CleanWhite Hugo Theme (https://themes.gohugo.io/hugo-theme-cleanwhite) by Huabing (https://zhaohuabing.com) | Star 804