MCP -> wiring diagram

Below is a minimal but complete “wiring diagram” + code snippets that let:

- a React/JS chat UI
- talk to your existing backend over the WebSocket `wss://backend.chatbuilder.com/events/listen`
- which forwards every user sentence to **your** Node orchestrator (the “bot”)
- that hosts an LLM (Anthropic or OpenAI) **and** an MCP client
- which calls an MCP **server** (also Node) that owns the Airtable CRUD helpers
- and finally ships the answer back the same chain.

No Claude Desktop, no stdio, everything is plain HTTP/SSE inside your own VPC.

---

1. Component map

Chat UI ⇄ WSS ⇄ Backend.chatbuilder.com ⇄ HTTP ⇄ Bot/orchestrator ⇄ SSE ⇄ Airtable-MCP-server
(React) (existing) (your Node service) (your Node MCP server)

- The **bot** keeps the LLM API key and the MCP client.
- The **MCP server** only knows Airtable PAT + base ID and exports tools like `airtable:select_records`, `airtable:create_record`, …
- Both services are Dockerised and scale horizontally.

---

2. Airtable MCP server (Node, SSE transport)

Install
mkdir airtable-mcp && cd airtable-mcp
npm init -y
npm install @modelcontextprotocol/sdk airtable dotenv
server.js
JavaScript
Copy

```
import { Server } from "@modelcontextprotocol/sdk/server/index.js";
import { SSEServerTransport } from "@modelcontextprotocol/sdk/server/sse.js";
import express from "express";
import Airtable from "airtable";
import "dotenv/config";

const app = express();
```

```
app.use(express.json());

const port = process.env.PORT || 8001;
const base = new Airtable({apiKey: process.env.AIRTABLE_PAT})
               .base(process.env.AIRTABLE_BASE_ID);

// 1. describe tools
const tools = [
  {
    name: "airtable:select_records",
    description: "List records from a table",
    inputSchema: {
      type: "object",
      properties: {
        table:   { type: "string" },
        filter:  { type: "string" },
        maxRecords: { type: "number", default: 10 }
      },
      required: ["table"]
    }
  },
  {
    name: "airtable:create_record",
    description: "Insert one record",
    inputSchema: {
      type: "object",
      properties: {
        table: { type: "string" },
        fields: { type: "object" }
      },
      required: ["table", "fields"]
    }
  }
];

// 2. instantiate MCP server
const server = new Server(
  { name: "airtable-mcp", version: "1.0.0" },
  { capabilities: { tools: {} } }
);
```

```
server.setRequestHandler("tools/list", async () => ({ tools }));
server.setRequestHandler("tools/call", async (req) => {
  const { name, arguments: args } = req.params;
  if (name === "airtable:select_records") {
    const recs = await base(args.table)
      .select({ maxRecords: args.maxRecords || 10, filterByFormula: args.filter || "" })
      .all();
    return {
      records: recs.map(r => ({ id: r.id, fields: r.fields }))
    };
  }
  if (name === "airtable:create_record") {
    const created = await base(args.table).create([{ fields: args.fields }]);
    return { id: created[0].id };
  }
  throw new Error("Unknown tool");
});

// 3. expose SSE endpoints
app.get("/sse", async (req, res) => {
  const transport = new SSEServerTransport("/message", res);
  await server.connect(transport);
});

app.post("/message", (req, res) => {
  const transport = SSEServerTransport.get(req.query.sessionId);
  if (transport) transport.handlePostMessage(req, res);
});

app.listen(port, () => console.log(`Airtable MCP listening on :${port}`));
```

.env
Copy

```
AIRTABLE_PAT=patXXXXXXXXXXX
AIRTABLE_BASE_ID=appXXXXXXXXXXX
```

Run

node server.js → http://localhost:8001/sse (SSE endpoint)

---

3. Bot/orchestrator (Node, hosts LLM + MCP client)

---

mkdir bot && cd bot
npm init -y
npm install @modelcontextprotocol/sdk axios dotenv express
bot.js
JavaScript
Copy

```
import { Client } from "@modelcontextprotocol/sdk/client/index.js";
import { SSEClientTransport } from "@modelcontextprotocol/sdk/client/sse.js";
import axios from "axios";
import express from "express";
import "dotenv/config";

const app = express();
app.use(express.json());

// 1. connect MCP client to airtable server
const mcp = new Client({ name: "chat-bot", version: "1.0.0" });
const transport = new SSEClientTransport("http://localhost:8001/sse");
await mcp.connect(transport);
const tools = await mcp.listTools();

// 2. small helper: talk to LLM
async function callLLM(messages) {
  const body = {
    model: process.env.LLM_MODEL,        // "claude-3-5-sonnet-20241022" or "gpt-4-turbo"
    messages,
    tools: tools.map(t => t.inputSchema ? { ...t, function: t.inputSchema } : t),
    tool_choice: "auto",
    max_tokens: 2000
  };

  const url = process.env.LLM_PROVIDER === "anthropic"
    ? "https://api.anthropic.com/v1/messages"
    : "https://api.openai.com/v1/chat/completions";
```

```
  const headers = process.env.LLM_PROVIDER === "anthropic"
    ? { "x-api-key": process.env.ANTHROPIC_KEY, "content-type": "application/json" }
    : { "authorization": `Bearer ${process.env.OPENAI_KEY}`, "content-type":
"application/json" };

  const { data } = await axios.post(url, body, { headers });
  return data;     // returns Claude or OpenAI shape
}

// 3. single HTTP endpoint that backend.chatbuilder.com will call
app.post("/handle_turn", async (req, res) => {
  const userSentence = req.body.text;          // comes from backend via HTTP
  const conversation = [{ role: "user", content: userSentence }];

  // first LLM call
  let llmResp = await callLLM(conversation);
  let assistantMsg = llmResp.content || llmResp.choices[0].message;

  // handle tool calls
  if (assistantMsg.tool_calls || assistantMsg.function_call) {
    const toolCalls = assistantMsg.tool_calls || [assistantMsg.function_call];
    for (const tc of toolCalls) {
      const name = tc.function?.name || tc.name;
      const args = JSON.parse(tc.function?.arguments || tc.arguments);
      const result = await mcp.callTool(name, args);
      conversation.push(assistantMsg);
      conversation.push({ role: "tool", tool_call_id: tc.id, content: JSON.stringify(result)
});
    }
    // second call with tool results
    llmResp = await callLLM(conversation);
    assistantMsg = llmResp.content || llmResp.choices[0].message;
  }

  const replyText = assistantMsg.content || assistantMsg.text || assistantMsg;
  res.json({ reply: replyText });   // goes back to backend.chatbuilder.com
});

app.listen(3000, () => console.log("Bot/orchestrator on :3000"));
```

.env
Copy

```
LLM_PROVIDER=anthropic        # or openai
ANTHROPIC_KEY=sk-ant-xxx
OPENAI_KEY=sk-xxx
LLM_MODEL=claude-3-5-sonnet-20241022   # or gpt-4-turbo
```

---

4. Glue inside backend.chatbuilder.com

You already have a WebSocket handler.
Add (pseudo):
JavaScript
Copy

```
// when a message arrives from UI
ws.on('message', async (data) => {
  const { text, userId } = JSON.parse(data);
  // forward to bot/orchestrator
  const { data: { reply } } = await axios.post(
    "http://bot-service:3000/handle_turn",
    { text, userId }
  );
  // send answer back to same websocket
  ws.send(JSON.stringify({ type: "bot_reply", text: reply }));
});
```

---

5. One-shot docker-compose for local dev

---

yaml
Copy

```
version: "3.8"
services:
```

```
  airtable-mcp:
    build: ./airtable-mcp
    ports: ["8001:8001"]
    env_file: ./airtable-mcp/.env
  bot:
    build: ./bot
    ports: ["3000:3000"]
    env_file: ./bot/.env
    depends_on: [airtable-mcp]
```

`docker compose up` → everything spins up, UI talks to your existing backend, backend forwards to bot, bot calls Airtable via MCP, answer flows back.

---

6. What you gained

- Chat UI ⇄ WSS stays untouched.
- Backend.chatbuilder.com only needs to **forward** text to the bot service; no Airtable keys, no LLM keys, no MCP logic.
- Airtable CRUD lives in its own container; expose extra tools (update, delete, linked tables, …) by editing only the MCP server.
- Swap Anthropic ↔ OpenAI by changing one env var.
- Add Google-Calendar MCP server on port 8002, register it in the bot startup loop—zero other changes.

Revision #2
Created 23 September 2025 04:17:11 by Admin
Updated 23 September 2025 04:26:01 by Admin