Backboard.io | Developer Integration Guide

Introduction:

This guide provides a structured roadmap for integrating backboard.io services. It is designed to bridge the gap between initial environment setup and production-ready implementation, establishing a consistent and efficient workflow for the entire engineering team.

Phase 1: Environment Setup

1. Account Setup

- **Sign Up:** Visit backboard.io.
- **Promo Code:** Use '**DEVSOC2026'** during registration.
- **API Access:** Navigate to **"Manage API Keys"** and generate a new key. Keep this key secure as it is required for all requests.

2. The Developer Workspace: API Testing with Postman

While your project code is the final product, **Postman** is the professional workspace where you test and refine your backend logic. It allows you to simulate requests and visualize responses instantly, ensuring your integration is optimized before you write a single line of frontend code.

How to run your first test:

Step 1: The Setup (Do this for all requests)

1. Open a new tab in Postman and set the method to **'POST'**.
2. Go to the **Headers** tab.
3. Under **Key**, type `X-API-Key`.
4. Under **Value**, paste your API Key.

Step 2: Testing the api endpoints

1. Create Assistant

- **URL:** `https://app.backboard.io/api/assistants`

Body: Select **raw** > **JSON** and paste

```
{
  "name": "YOUR_BOT_NAME",
  "system_prompt": "Describe your bot"
}
```

- **Note:** Copy the `assistant_id` from the response, and save it.

2. Create Thread

- **URL:** `https://app.backboard.io/api/assistants/YOUR_ASSISTANT_ID/threads` *(Replace `YOUR_ASSISTANT_ID` with the ID from Step 1)*
- **Body:** Keep it empty `{}` but ensure it is set to **raw** > **JSON**.
- **Note:** Copy the `thread_id` from the response.

3. Send Message

- **URL:** `https://app.backboard.io/api/threads/YOUR_THREAD_ID/messages` *(Replace `YOUR_THREAD_ID` with the ID from Step 2)*
- **Body:** Select **form-data** (not JSON for this specific endpoint) and add these keys:
  - `content`: `Tell me about Canada in detail.`
  - `stream`: `false`
  - `memory`: `Auto`
- **Send:** Hit send, and your answer will appear in the response box.

3. Local Installation

Run the following in the terminal in your project directory to initialize your project:

```
npm init -y
npm install express backboard-sdk
```

Phase 2: Key terms

To build effectively on Backboard, understand this hierarchy:

1. **Assistant:** The "Brain." Configured with specific instructions and tools. Each assistant has a “name” (is given by you, to refer to it) and a “assistantId” (generated, and is unique to every assistant).
2. **Thread:** A specific “Conversation” with an assistant. It may/may not save important points in the conversation (based on the mode selected by you, during assistant creation)
3. **Message:** The "Interaction." The actual data exchange between the user and AI.

Phase 3: Implementation Snippets

1. Basic Interaction

This snippet creates an assistant and starts a conversation. By setting memory: 'Auto', the assistant will intelligently remember user facts across different threads.

```
import { BackboardClient } from 'backboard-sdk';

async function main() {
  const client = new BackboardClient({ apiKey: 'YOUR_API_KEY' });

  // Create an Assistant
```

```
  const assistant = await client.createAssistant({
    name: 'NAME_YOUR_ASSISTANT',
    system_prompt: "Define your assistant's behavior and personality"
  });

  // Create a Thread (The conversation session)
  const thread = await client.createThread(assistant.assistantId);

  // Send a message
  const response = await client.addMessage(thread.threadId, {
    content: "Hello! Tell me a fun fact about space.",
    llm_provider: "openai", // Default: openai
    model_name: "gpt-4o",   // Default: gpt-4o
    memory: 'Auto',         // Enables persistent memory
    stream: false
  });

  console.log("AI Response:", response.content);
}

main().catch(console.error);
```

2. Tool Calling (Custom Logic)

Enables the assistant to interact with your project's backend. This "handshake" allows the AI to trigger functions you define locally.

```
// 1. Define the tool
const tools = [{
  type: 'function',
  function: {
    name: 'YOUR_TOOL_NAME',
    description: 'Explain when the AI should use this tool',
    parameters: {
      type: 'object',
      properties: {
        DATA_FIELD: { type: 'string', description: 'Input required from AI' }
      },
      required: ['DATA_FIELD']
    }
  }
}];

// 2. Execute and handle the tool call
```

```
const response = await client.addMessage(thread.threadId, {
  content: "User query requiring a tool",
  stream: false
});

if (response.status === 'REQUIRES_ACTION' && response.toolCalls) {
  const toolOutputs = response.toolCalls.map(tc => {
    const args = tc.function.parsedArguments;

    // YOUR CUSTOM LOGIC HERE (e.g., Database fetch)
    const resultData = { info: "Result based on " + args.DATA_FIELD };

    return { tool_call_id: tc.id, output: JSON.stringify(resultData) };
  });

  const finalResponse = await client.submitToolOutputs(
    thread.threadId, response.runId, toolOutputs
  );
  console.log(finalResponse.content);
}
```

3. Document Intelligence & Web Search

Give your AI access to local files and live internet data.

- **For Web Search:** Add web_search: 'Auto' to your addMessage parameters.
- **For Documents:**

JavaScript

```
// Upload and link document to assistant
const doc = await client.uploadDocumentToAssistant(assistant.assistantId,
'file.pdf');

// Poll status until 'indexed'
const status = await client.getDocumentStatus(doc.documentId);
if (status.status === 'indexed') {
  console.log("Assistant can now answer questions about the file.");
}
```

Phase 4: Troubleshooting

Status	Meaning
IN_PROGRESS	AI is still processing/thinking. Wait for completion.
REQUIRES_ACTION	AI is waiting for you to run a **Tool/Function**.
FAILED	Check your API Key or JSON body format.
401 Unauthorized	Your X-API-Key header is missing or incorrect.