Captain’s Log Custom GPT

Overview

The Captain’s Log custom GPT is a personalized note-taking and logging application designed specifically for a single user. It enables the user to log and manage day-to-day tasks, interactions, and ideas using GPT’s input parsing capabilities to extract structured data. Vercel serves as a middleware to validate and format this data, ensuring seamless and secure storage in Supabase. The application is tailored to the user’s workflow, allowing for efficient logging, follow-up tracking, and data retrieval based on their unique needs.

User Context  
The user, Luke Pickard, is a 40-something IT Director at NASG, currently based at the TN2 location in Portland, TN. Married with two step-kids, Luke was born in Oxford, MS, and spent a decade in Chicago before moving to Nashville, TN, in 2016 to work for a market research company. After seven years in that role, he earned his MBA in 2023 and was promoted to Senior Security Director. In June 2024, Luke transitioned to NASG as IT Director, embracing a role with higher pay and growth potential within the organization. Reporting directly to CFO Rick Marsh, a financially driven and stern supervisor with a close relationship to CEO Michael Haughey, Luke is dedicated to advancing within NASG while leveraging his extensive experience to support the company’s strategic goals. Recognizing the challenges of tracking and managing the many tasks, interactions, and responsibilities in his demanding role, Luke plans to use Captain’s Log to streamline his workflow, ensure nothing falls through the cracks, and boost his overall productivity.  
  
As IT Director at NASG, I manage a team of four employees, each based at specific locations to support our operations. My team includes three Systems Analysts: James, located at ON1, who oversees IT operations for ON1, ON2, MI1, and OH1; Hector Martinez Lopez, based at QR1, who manages QR1, QR2, OH3, and OH2; and Andrew Hannah, stationed at TN2, who supports TN1, TN2, and TN3. Andrew’s father, David Hannah, also works for NASG in TN2. Additionally, Tomo Kokuryo, a Business Intelligence Manager, is also located at TN2. Together, they share responsibility for IN1 and OH4 as these facilities prepare to close by the end of 2024. My leadership philosophy emphasizes motivating my team, treating them professionally, and encouraging their growth, while constructively addressing flaws to help them improve. My goal is to guide them toward achieving their career aspirations while fostering both personal and professional development.

NASG

North American Stamping Group (NASG) is a leading manufacturer of metal stampings, welded assemblies, and tooling solutions for the automotive industry, with facilities across the United States, Canada, and Mexico, each identified by a unique three-digit code. In the U.S., NASG operates in Portland, TN (TN1 and TN2), with TN2 serving as the corporate headquarters featuring 235,000 sq. ft. of manufacturing space and 16 presses ranging from 400 to 1500 tons. The company also operates TN3 in Pulaski, TN; Ada, OH (OH1); Ridgeville Corners, OH (OH2), home to a 55,000 sq. ft. tooling and automation facility; Paulding, OH (OH3), a 110,000 sq. ft. wire forming and welding-focused facility; and Bryan, OH (OH4), which will close by the end of 2024. NASG also maintains a sales and engineering hub in Farmington Hills, MI (MI1).

In Canada, NASG operates ON1, a 254,000 sq. ft. manufacturing plant in Ontario specializing in progressive stamping and welding, and ON2, a 22,000 sq. ft. tech center focused on stamping tool design and build. In Mexico, NASG has QR1, a 245,000 sq. ft. facility with presses up to 2000 tons offering progressive stamping and robotic MIG welding, and QR2, a 65,000 sq. ft. plant opened in 2022 that focuses on welded assemblies and advanced automation. These facilities support critical automotive markets such as seating, ride control, NVH, emissions, and chassis and structural components. Committed to innovation, efficiency, and quality, NASG leverages advanced technology and a skilled workforce to deliver medium-to-high-volume components to Tier I suppliers and OEMs, maintaining its position as a trusted partner in the industry.

Database, Relationships and Data Flow

The Captain’s Log application leverages a relational database in Supabase to efficiently manage and retrieve user logs, contacts, and companies. At its core, the LogEntries table serves as the primary repository for event data, including log type (e.g., encounter, meeting, note, email, task or other), keywords, follow-up flags, timestamps, and detailed text. The table’s id acts as the primary key, uniquely identifying each log entry. All things marked as followups should be labeled as a task.

To establish relationships between logs, contacts, and companies, the database uses two many-to-many relationship tables: LogEntryContacts and LogEntryCompanies. The LogEntryContacts table includes a composite primary key made up of logentryid and contactid. The logentryid is a foreign key referencing the id in the LogEntries table, while contactid is a foreign key linking to the id in the Contacts table. Similarly, the LogEntryCompanies table also uses a composite primary key of logentryid and companyid. Here, logentryid is a foreign key referencing the LogEntries table, and companyid links to the id in the Companies table. These relationships allow logs to be associated with multiple contacts and companies, ensuring flexibility in managing complex interactions.

The Contacts table centralizes information about individuals, including their first name, last name, email, and optional phone number. Each contact is uniquely identified by the id as its primary key, and the companyid field serves as a foreign key linking contacts to the Companies table. The Companies table organizes organizational data, including company names, locations (city, state, and zip), and phone numbers, with its id acting as the primary key.

The data flow begins with user input, which GPT processes to extract structured data such as event type, keywords, contacts, and companies. Vercel acts as middleware, validating the extracted data and ensuring consistency before sending it to Supabase for storage. Log entries are created in the LogEntries table, while relationships to contacts and companies are established in the LogEntryContacts and LogEntryCompanies tables using foreign keys. This structure ensures that logs are always connected to relevant entities.

For data retrieval, the application queries the LogEntries table and applies joins on the LogEntryContacts, Contacts, LogEntryCompanies, and Companies tables as needed. For example, a query for pending follow-ups filters logs where the followup flag is set to TRUE, while foreign key relationships ensure associated contacts and companies are included in the results. Similarly, keyword filtering or searching by specific individuals leverages the relationships between tables to return comprehensive results.

This relational database structure ensures that data integrity is maintained through the use of primary keys and foreign keys. The current design supports efficient management of logs, allowing for seamless storage, linking, and retrieval of detailed interactions and tasks.

API Programing (.js and .json)  
  
The Captain’s Log backend API is organized into a structured file system that handles operations for companies, contacts, and logentries while ensuring secure, efficient, and modular functionality. The api folder serves as the central hub for all endpoints, with each resource—companies, contacts, and logentries—organized into its respective subfolder. Each subfolder contains two primary files: [id].js for operations involving a specific resource by its id and index.js for general operations, such as creation, retrieval, updates, and deletions. Authorization is enforced across all

routes using an API key, which is validated against the authorization header and compared to the environment variable OPENAI\_KEY. All invalid attempts return a 401 Unauthorized error, ensuring secure access.

In the companies folder, the [id].js file handles PATCH and DELETE requests for individual companies. The PATCH method updates fields like name, city, state, zip, phone, and country, requiring at least one valid field to proceed. The DELETE method removes a specific company by its id and returns a confirmation message. The index.js file supports the POST, GET, PATCH, and DELETE methods. The POST method creates a new company record, validating the presence of the name field and filtering undefined fields before inserting data into the companies table. The GET method retrieves all companies or a specific company by id. It also supports the includeLogs query parameter, which fetches logs associated with a company by joining the logentrycompanies table with the logentries table. The PATCH method updates fields for a specific company, while the DELETE method ensures relational integrity by cascade deleting related records in the logentrycompanies table before removing the company itself.

The contacts folder mirrors the structure of companies with [id].js and index.js. The [id].js file supports PATCH for updating fields like firstname, lastname, email, and phone, ensuring at least one field is provided for updates, and DELETE for removing a specific contact. The index.js file supports POST, GET, PATCH, and DELETE operations. The POST method dynamically handles contact creation, reconstructing the firstname and lastname fields from a combined name input if needed. It also infers the company name from an email domain using a helper function inferCompanyFromEmail, and if no company exists, it creates a new company record in the companies table. The GET method retrieves all contacts, optionally filtered by name or by a company association using the company query parameter. It also supports fetching logs related to a contact using the includeLogs parameter, joining the logentrycontacts and logentries tables. The DELETE method allows for removal of contacts by id or name, performing a cascade delete in the logentrycontacts table to maintain data integrity.

The logentries folder handles operations for logging events and associating them with contacts and companies. The index.js file supports GET, POST, PATCH, and DELETE methods. The GET method retrieves log entries, optionally filtering by id, and includes relationships to contacts and companies by joining the logentrycontacts and logentrycompanies tables. The POST method creates a new log entry, extracting fields like logtype, keywords, text, and followup. It validates keywords to ensure they are single words without spaces and links the log entry to contacts and companies by inserting records into the logentrycontacts and logentrycompanies tables. The PATCH method updates a log entry’s fields and resets associations with contacts or companies if provided. The DELETE method cascade deletes related records in the logentrycontacts and logentrycompanies tables before removing the log entry.

The validateCompanies.js file provides helper functions to validate and manage company entries dynamically. The extractAndValidateCompanies function identifies potential company names from log text and keywords, excluding stopwords like “meeting” or “task.” It checks for existing company records and prepares new entries if needed. The createCompanyIfMissing function inserts a new company into the companies table with optional details such as city, state, zip, and phone. These helper functions streamline the process of associating logs with company records.

The utils/supabase.js file initializes the Supabase client using environment variables SUPABASE\_URL and SUPABASE\_KEY. This centralized setup avoids repetitive client initialization across files, ensuring consistency and easier maintenance.

The vercel.json file configures the Vercel platform for deployment and routing. It specifies the build process, targeting all JavaScript files under the api folder as serverless functions using the Node.js runtime. The routing configuration maps incoming requests to their respective API endpoints based on the x-subdomain header. Requests with the header x-subdomain: logentries are routed to /api/logentries, those with x-subdomain: contacts are routed to /api/contacts, and requests with x-subdomain: companies are routed to /api/companies. All routes support the HTTP methods GET, POST, PATCH, and DELETE. Additionally, a fallback rule ensures that any other requests targeting /api/ are passed directly to their respective handlers.

Together, the API structure, utility functions, and routing configuration provide a robust backend for managing companies, contacts, and log entries. The modular design ensures clean separation of responsibilities, secure API access through key validation, and efficient database interactions with Supabase. This setup supports relational integrity, dynamic data handling, and a flexible architecture that is easy to scale and maintain.

Custom GPT Actions  
The Captain’s Log custom GPT actions utilize well-defined API endpoints to interact with logentries, companies, and contacts through the provided schemas. These actions enable the GPT to log tasks, manage relationships between logs, contacts, and companies, and maintain data consistency across entities.

The LogEntries API is hosted at https://caplog-lukeamotion.vercel.app and provides full CRUD operations. The POST method (createLogEntry) allows the creation of new log entries by specifying a logtype (e.g., “Call,” “Meeting,” or “Note”), keywords (tags), followup status, text content, and optional associations through contactids and companyids. Required fields include logtype, keywords, and text. The GET method (getLogEntries) retrieves all log entries, with optional filters for keywords and followup status. It also includes the ability to retrieve associated contacts and companies by setting the includeRelationships query parameter to true. The PATCH method (updateLogEntry) updates specific fields in a log entry, such as logtype, keywords, followup, text, and updates its relationships with new contactids and companyids. The DELETE method (deleteLogEntry) removes a log entry and its associations, requiring the id parameter for identification.

The Companies API is hosted at https://companies.caplog.lukeamotion.com and facilitates management of company records and their associations. The POST method (createCompany) creates a new company and allows optional linking of contactids and logentryids to establish relationships. Required fields include name, while optional fields like city, state, zip, country, and phone can provide additional details. The GET method (getCompanies) retrieves all companies or filters results by name. By setting query parameters like includeContacts or includeLogs to true, the response can include associated contacts and log entries. The PATCH method (updateCompany) updates specific company fields, such as name, city, state, zip, country, or phone, and allows updating its relationships with new contactids and logentryids. The DELETE method (deleteCompany) removes a company and its associations, requiring the id query parameter for identification.

The Contacts API, hosted at https://contacts.caplog.lukeamotion.com, focuses on managing contacts and their relationships with logs and companies. The POST method (createContact) creates a new contact using name and email as required fields. If no company is provided, the API will infer the company name from the contact’s email domain (e.g., “microsoft.com” → “Microsoft”). If inference fails, the contact is created without a company association. The GET method (getContacts) retrieves all contacts, with optional filters for name, companyid, or logentryid to narrow down results based on relationships. The PATCH method (updateContact) updates a contact’s details, such as name, email, phone, companyid, and associated logentryids. The DELETE method (deleteContact) removes a contact and its associations with logs, requiring the id query parameter for identification.

These custom GPT actions enable seamless communication with the Captain’s Log backend, facilitating the creation, retrieval, update, and deletion of logs, companies, and contacts. By leveraging query parameters and structured request bodies, the GPT can dynamically filter data, infer relationships, and maintain referential integrity across entities. This allows users to efficiently manage their tasks, interactions, and relationships in an organized and scalable way.

GPT Instructions  
The Captain’s Log GPT serves as an interface to manage tasks, interactions, and follow-ups via the CapLog API. The default endpoint is /api/logentries/ unless explicitly directed otherwise, and all API calls must include the /api prefix when interacting with any endpoint. Natural language directives can switch between endpoints, such as /companies/ or /contacts/, enabling seamless navigation and data management.

When creating or managing log entries, contacts, or companies, the GPT intelligently links entities. Contacts are automatically linked to companies based on their email domains. If a new company is mentioned in a log entry, the GPT checks the /companies/ endpoint to confirm its existence. If the company does not exist, it is automatically created with available headquarters details such as City, State, ZIP, and Phone. If these details are not provided, the GPT proceeds with the company name only. Newly created companies are automatically linked to the corresponding log entry, ensuring relational consistency. In cases where multiple companies match an email domain, the GPT requests clarification from the user. The GPT also infers details and identities from context where possible; if further clarification is needed, it prompts the user for additional information to maintain accuracy.

The user, Luke Pickard, is the IT Director for North American Stamping Group (NASG) based at the NASG TN2 plant in Portland, TN. References to “our” refer to NASG as a whole, not a specific team. NASG locations are identified using plant codes, with NASG (TN1) referring to Portland, NASG (TN2) being Luke’s location, NASG (OH1) and NASG (OH3) mapping to Paulding, and NASG (ON1) located in Woodstock. For example, if the user asks for “the number to our TN1 plant,” the GPT understands it refers to NASG (TN1) in Portland.

Enhanced features further streamline task and follow-up management. At the start of each day, when the user logs the first entry, the GPT automatically checks for tasks due that day (followuprequired=true) and any other open follow-ups, sorting them by urgency. For instance, the GPT might output a morning summary that highlights tasks like “Rick Marsh: Submit revised budget (Due: 12/18/2024)” and “NASG (OH3): Confirm equipment repairs (Due: 12/18/2024).” Other open tasks, such as “Justin Briley: Submit vendor quotes (Due: 12/20/2024)” or “NASG (TN1): Resolve IT downtime issues (No due date),” are also displayed for quick prioritization. Similarly, when asked “What tasks should I focus on?”, the GPT prioritizes open tasks by due date, separating overdue tasks such as “NASG (OH2): Resolve production outage from 12/15/2024” from tasks that are due soon, like “Rick Marsh: Submit revised budget (Due: 12/18/2024).”

To map locations seamlessly, the GPT associates NASG plant codes with their corresponding cities. For example, NASG (OH3) maps to Paulding, NASG (TN1) maps to Portland, NASG (TN2) represents the user’s location, and NASG (ON1) corresponds to Woodstock. A query like “What’s happening at our Paulding plant?” results in a summary specific to NASG (OH3), such as highlighting open tasks like “Confirm equipment repairs with the maintenance team” or referencing recent notes like “Discussed downtime during the last plant inspection on 12/05/2024.”

The GPT also assists with email drafting and review. When drafting emails to executives, it adapts tone and style based on the recipient’s role. For Rick Marsh, the CFO, the GPT produces clear and concise updates with actionable details in a professional but direct tone. For Michael Haugey, the CEO, it ensures formal, solution-focused communication that efficiently highlights key points. For Mike Rigsby, the CTO, it uses technical precision, emphasizing results or technical insights. For example, when asked to “Draft an email to Rick Marsh about the revised budget,” the GPT might generate: “Subject: Follow-Up on Revised Budget. Hi Rick, I’m following up on the Q1 budget discussion. Could you send over the revised budget by 12/18/2024? Let me know if you need additional information. Thanks, Luke.”

Additionally, the GPT reviews email drafts to ensure executive presence, professionalism, grammar, and formatting. It refines subject lines to improve clarity, adjusts the tone to suit the recipient, and polishes the call to action. For example, when reviewing an email to Michael Haugey, it might suggest changes like “Refined the subject line to highlight urgency: ‘Strategic Update: Revised Budget for Q1’” and “Polished the body to ensure executive-level clarity and focus on outcomes.”

The GPT assumes that all actions apply to the /api/logentries/ endpoint unless otherwise directed. It tracks flagged tasks (followuprequired=true) as open and unflagged tasks as closed, prioritizing due and overdue follow-ups. Log entries are automatically tagged based on keywords, including plant codes like TN1, TN2, and OH3, as well as their corresponding cities such as Portland and Paulding. The GPT also provides concise, clear summaries for tasks, notes, and reports upon request, ensuring that the user has quick access to critical information to stay organized and productive.