ChatGPT

LLM-Driven GDPR Compliance Q&A Graph – Technical Brief

Overview and Goals

This project aims to build an **interactive chatbot UI** that guides a user through a series of questions about their organization's GDPR practices, dynamically builds a **knowledge graph** from the answers, and then provides personalized GDPR guidance. Unlike static questionnaires, this approach uses a Large Language Model (LLM) to adapt questions to the user's context in real time [1]. The goal is an MVP (Minimum Viable Product) that showcases the end-to-end experience: asking up to **10 questions** about GDPR compliance, capturing the information in a semantic graph, confirming the captured info with the user, and finally outputting tailored recommendations. This MVP will be entirely client-side (no server storage), leveraging the user's browser (and local storage) to store state and an LLM API (e.g. OpenAI GPT-4) for intelligence. The focus is **specifically on GDPR mapping** in this phase (building the user's GDPR compliance profile graph); broader persona graph integration can be added later as needed.

Key objectives include:
- Providing a **chat-style interface** where the system (LLM) asks the user GDPR-related questions in a friendly, conversational manner.
- Dynamically constructing a **graph of nodes and edges** representing the user's context and answers (GDPR practices), updated after each question [1] [2].
- Ensuring the LLM uses the growing graph to inform subsequent questions, creating a personalized Q&A flow (max 10 Qs) rather than a fixed survey [3].
- Including a **confirmation step** where the system shows the user what information has been captured (the graph's contents) and asks for verification (Yes/No) [4].
- Generating a **final guidance report** or recommendations based on the completed graph (e.g. highlighting GDPR compliance gaps and next steps), written in an accessible, helpful tone.

This brief describes the technical design and prompts needed for an LLM to "vibe code" the MVP – meaning the LLM can generate the UI code and logic from this specification, without us writing the low-level code ourselves. The **tone and UX** should remain friendly and engaging throughout, as if the user is conversing with a helpful assistant.

Workflow Summary

The end-to-end interaction will follow these stages:

1. **Initial Question:** The conversation starts with the system asking a broad, opening question to establish context. For example: *"What industry is your company in?"* [2]. This can be a hardcoded first question or generated by the LLM. The UI displays this question (ideally streaming the text as it comes from the LLM for a natural feel).
2. **User Answer:** The user enters their answer (e.g. "Healthcare industry"). The front-end collects this input.

3. **Graph Update (LLM Call):** The user's answer is sent to the LLM with a prompt to **update the knowledge graph**. The LLM returns a JSON representation of the updated graph that now includes the new facts from the answer. For instance, after "healthcare" as industry, the graph gains a node for *Healthcare* and links it to the user's persona [2]. Under the hood, the LLM performs entity extraction on the free-form answer to structure it: *"Healthcare"* might be recognized as an Industry node, and an edge `Persona --industry--> Healthcare` is added. Using an LLM for this extraction is crucial for flexibility – it can parse arbitrary text into structured data in a scalable way [5]. The UI will parse the returned JSON and update any client-side graph state.
4. **Next Question Generation (LLM Call):** After updating the graph, the system needs the next question. The front-end calls the LLM again, providing the current graph (in JSON) and a prompt to generate the **next best question** to ask. The LLM uses the graph context to craft a relevant follow-up question [6]. For example, knowing the industry is healthcare, it may ask a question focusing on a GDPR aspect relevant to healthcare (perhaps about handling health data). The conversation is adaptive: the LLM plans the dialogue to gather all needed GDPR info, rather than following a preset list [3]. We can guide the LLM by supplying a list of **information targets** (key GDPR compliance areas) that need to be covered, and it will decide the order and phrasing of questions to fulfill those objectives [3].
5. **Display Next Question:** The UI shows the next question to the user (again, using a friendly tone and possibly streaming text). The cycle then repeats: user answers, LLM updates graph, LLM produces another question. Each iteration enriches the knowledge graph with new nodes/edges (for example, adding nodes like "Has Data Protection Officer" with yes/no value, "Uses cloud storage", "Encryption in place", etc., as these facts emerge from answers). This loop continues until **approximately 10 questions** have been asked and answered, or until the LLM indicates that all target information has been gathered. (For MVP, we'll simply stop at 10 questions to enforce scope.)
6. **Confirmation Step:** After the Q&A loop, the system will compile a summary of the user's provided information – essentially reflecting the content of the graph back to the user for verification [4]. This can be presented as a list of bullet points or a short paragraph, e.g.: *"You indicated: (1) Your industry is healthcare, (2) You have a Data Protection Officer appointed, (3) You handle EU customer data and regularly conduct privacy training, … Is all this correct?"*. The user is prompted with a simple **"Yes/No" confirmation** asking if the captured profile is accurate [4]. (For this MVP, we assume the user confirms "Yes." Handling a "No" could involve allowing corrections, but that is an extension for later – for now we focus on the happy path.) The confirmation step is important for accuracy and user trust: it ensures the graph isn't silently accumulating possibly incorrect assumptions – the user gets to validate the data [7].
7. **Guidance Generation (Final LLM Call):** Once confirmed, the final step is to provide **personalized GDPR guidance or recommendations**. The front-end calls the LLM one more time with a prompt containing the completed knowledge graph and an instruction to output a helpful guidance report. The LLM will produce a response like a summary of the user's GDPR compliance status and recommended next steps to improve or ensure compliance. For example, it might say: *"Based on what you've told me, your organization has a Data Protection Officer and training programs, which is great. However, it looks like you might not yet have a formal process for handling data subject access requests – implementing a clear DSAR process would be an important next step. Additionally, since you work in healthcare, ensure that all health data is encrypted and access is logged. …"*. This response should be in a friendly, advisory tone, not judgmental, and should reference the information in the graph. Because the user helped build the graph (and confirmed it), they are more likely to trust these final recommendations [7].
8. **Display Guidance:** The UI presents the LLM's guidance to the user, concluding the interaction. Optionally, the interface might allow the user to download this report or restart the process. In

future iterations, this guidance could be accompanied by a visual representation of their compliance graph or links to resources, but for the MVP a textual output suffices.

Throughout the above workflow, **no server-side state** is maintained – all data (the graph JSON, the conversation context) lives in the client (browser). Each LLM call is made directly from the front-end to the LLM API. The user's API key (for the OpenAI service, for instance) will be stored in the browser (e.g. in `localStorage`) and used for these calls. This keeps the architecture simple and within the client, acknowledging that sensitive info is being sent to the LLM API but nothing is permanently stored on our servers.

Knowledge Graph Representation

At the core of the system is the **knowledge graph** that accumulates the user's answers in structured form. We need a data format that the LLM can reliably output and the front-end can easily parse. Two possible approaches are:

- **Custom JSON Schema:** A straightforward JSON structure with lists of nodes and edges. For example, we might represent the graph as:

```
{
  "nodes": [
    {"id": "persona", "type": "Organization", "name": "UserCompany"},
    {"id": "industry_healthcare", "type": "Industry", "name":
"Healthcare"}
  ],
  "edges": [
    {"source": "persona", "target": "industry_healthcare", "relation":
"industry"}
  ]
}
```

  In this example, after the user answered industry = healthcare, we have a node for the persona (the user's organization or persona node) and a node for the healthcare industry, with an edge indicating the relationship. We could choose a simple naming scheme for node IDs and a limited set of relation types (like `"industry"`, `"has_dpo"`, `"uses_cloud"`, `"conducts_training"`, etc. as needed for GDPR facts). The LLM would output the updated JSON each time the graph grows. This custom JSON is flexible and probably easier for the LLM to output correctly (less risk of strict syntax errors than a fully context-heavy format). The downside is it's not inherently semantic to outside tools – it's a format we define just for this app.

- **JSON-LD (Linked Data):** A semantically rich format that could align with standard vocabularies (like schema.org or GDPR ontologies). JSON-LD would allow us to attach context and types to nodes using IRIs. For instance, we might declare the persona as an `@type: Organization` with an `"industry": "Healthcare"` property, or use a custom GDPR vocabulary for things like Data Protection Officer. Modern LLMs can indeed produce JSON-LD if instructed [8]. For example, ChatGPT can output a JSON-LD document with a given context and types (the cited example shows an LLM generating a glossary in JSON-LD format) [8]. Using JSON-LD would make our graph immediately compatible with semantic web tools and easier to extend (it doubles as documentation of the ontology). However, it can be verbose and there's a higher

chance the LLM might make small formatting mistakes or not adhere perfectly to the context without careful prompt design.

For this MVP, a **pragmatic choice** is to use a **simplified JSON schema** for the graph that the LLM can manage easily. We can design it to be somewhat semantic (by including a `"type"` field for each node and using human-readable relation labels) to capture ontology-like information without full JSON-LD complexity. The LLM's graph-update prompt will explicitly instruct it on the JSON format to use, including the exact keys and structure expected, to minimize errors. As we iterate, we could move to a more formal JSON-LD if needed, especially if integrating with external knowledge bases. The important point is that each answer from the user will result in a structured update – either adding a new node (with a type and value) and an edge linking it to another node, or updating an attribute on an existing node. The graph thus grows **incrementally** with each interaction [1]. By the end of the Q&A, the graph contains all the key data points about the user's GDPR compliance posture (e.g. industry, presence of DPO, data inventory status, training, breach response plan, etc.) linked to the central persona.

We should also define an initial state for the graph. Likely, it starts with a single node representing the **user or organization** (the entity whose GDPR practices we are capturing). For example, the initial graph JSON could be as simple as:

```
{"nodes": [ {"id": "persona", "type": "Organization", "name":
"UserOrganization"} ], "edges": []}
```

The first question's answer will then add the first new node and edge to this graph. Having this initial node (persona) gives a reference point to attach all subsequent information. (In a more complex persona graph system, this might be pre-populated with some known info, but for now it's essentially empty aside from identity [1].)

LLM Prompt Design for Each Phase

We will use separate LLM API calls for different functions in the workflow. Each call will have its own **system prompt** (defining the role and behavior of the AI) and **input** (either user input or data like the current graph). Below are the main prompt schemas needed for each phase, along with the expected output:

- **1. Next Question Generation Prompt:** This prompt instructs the LLM to act as a *question generator* given the current state of knowledge. It will be used at the start (with a nearly empty graph) and then after each answer. We want the LLM to produce a single question for the user, in natural language, seeking a piece of GDPR-related information that has not been captured yet.

System Prompt Role: e.g. *"You are a helpful assistant interviewing a user to assess their GDPR compliance practices. Your goal is to figure out the next question to ask to gather missing information, based on what we already know. Always return the next question as a short, clear, and friendly question for the user, and nothing else."* We will include guidance about tone (friendly and professional) and format (just the question text).

Input: We provide the current graph (as JSON) and possibly a brief list of target topics remaining. For example, we might maintain a list like `["Industry","Company Size","Data Protection Officer","Data Inventory","Third-Party Data Sharing","Breach Response","Training","Data Subject Requests","Retention Policy","Encryption"]` – 10 items for 10 questions – and as each is addressed, remove it. The prompt can say: *"These are the*

information targets to cover: [list]. The graph below shows what we have so far. Based on this, generate the next question to address one of the remaining topics." This aligns with the method of giving the LLM a set of info objectives and letting it plan the conversation [3]. The current graph JSON will also be provided so the LLM knows which facts are already known (so it doesn't ask redundantly).

Output: The LLM's assistant answer will be a **single question sentence** (or a couple of sentences if needed for clarity) addressed to the user. For example: *"Do you have a designated Data Protection Officer (DPO) in your organization?"*. We will ensure the LLM does **not** include any JSON or additional text in this response – just the question string to display. The front-end will take this output and render it in the chat interface. (If using streaming, we'll stream the tokens to show the question appearing as the assistant "typing".) The LLM is effectively performing dialogue planning here, using the graph context to ask relevant questions [6].

- **2. Graph Update Prompt (After User Answer):** Once the user answers a question, we need to update our graph. This prompt makes the LLM act as a *knowledge graph builder*. It takes the user's latest answer and the current graph, and returns an updated graph JSON.

System Prompt Role: e.g. *"You are an AI that updates a JSON knowledge graph of a company's GDPR compliance based on new information. You will be given the current graph and the user's latest answer. Extract key facts from the answer and modify the graph JSON to include those facts as new nodes or edges. Only output the JSON structure of the updated graph, with no explanatory text."* We will define how to represent certain common structures. For instance, if the question was about having a DPO and the user answers "Yes, we have a DPO named Alice," the prompt should lead the LLM to add a node like `{"id": "dpo", "type": "Role", "name": "Data Protection Officer"}` (or possibly an attribute on the persona node like `"has_dpo": true` depending on our chosen schema) and perhaps even capture the name "Alice" if we want to store it (could be a property of the DPO node). We have to clearly instruct the LLM on how to handle *yes/no* type answers (maybe add a boolean attribute or a node that indicates presence of something) versus descriptive answers (add new concept nodes). Since GDPR topics can include binary and descriptive data, our schema should accommodate both. We might lean on adding nodes for both cases to keep it consistent (e.g., a yes could be represented by adding a node like `"Data Protection Officer: Yes"` or a boolean property). The exact design will be communicated in the prompt to the LLM.

Input: The prompt will include the **current graph JSON** and the **user's answer text**. For clarity and token limits, we can provide the graph in a concise form. The LLM will then output the revised JSON. We may have to remind the LLM not to lose existing nodes or edges – just to append or update. Also, instruct it to maintain proper JSON syntax (no trailing commas, etc.). We can include an example in the system prompt: e.g., *"If the current graph is X and the user says Y, the updated graph should be Z"* as a demonstration.

Output: Pure **JSON** (or JSON-LD) structure representing the new graph. The front-end will treat this as the authoritative state of the graph going forward. We'll parse it (e.g., using `JSON.parse` in JavaScript) and possibly validate it. This step effectively uses the LLM to do entity extraction and classification from the answer [5] [6]. For example, if the user mentioned *"We use Amazon S3 to store backups"* in an answer to a question about data storage, the LLM might add a node for *"Amazon S3 (Cloud Storage)"* and link it with an edge like `uses_cloud_service` to the persona. The use of an LLM here is what makes this scalable and flexible, as it can understand nuanced text and map it to our graph schema without rigid rules [5].

- **3. Confirmation Summary (Optional Prompt):** For the confirmation step, we have two approaches. The simpler approach is to **generate the summary on the client side** by reading

the graph data and formatting it into a human-readable list (as described earlier). This avoids an extra LLM call. However, we might also craft a prompt to let the LLM summarize the graph in a natural language paragraph, which could be more elegant. If we choose to use the LLM:

System Prompt Role: e.g. *"You are a summarizer that reviews a knowledge graph of GDPR data and explains it in a few sentences."* We could provide the graph JSON and ask for a summary of key points in second person ("you have… you do…"). But given we want to keep it simple and avoid potential LLM verbosity/mistakes, the MVP will likely **not** call the LLM here. Instead, the front-end will iterate through known nodes/edges and create a bullet list or sentence summary. For example, for each key node (excluding the root persona), we can create a sentence: *"Your industry is Healthcare," "You have a Data Protection Officer," "You use Amazon S3 for backups," etc.* assembling the list of facts. This is straightforward to implement with our structured data.

Output: Either a textual summary (if LLM was used) or a generated list. The UI will present this along with a question *"Is this information correct?"* and Yes/No buttons. The design should make it easy for the user to see all collected points at a glance (perhaps as a checklist).

- **4. Guidance Generation Prompt:** This prompt instructs the LLM to act as a *consultant or advisor*, producing GDPR guidance based on the user's profile (the graph).

System Prompt Role: e.g. *"You are an expert GDPR consultant AI. Given a structured profile of a company's GDPR compliance (below), provide a helpful set of recommendations and guidance. Address the user in a friendly, professional tone. Highlight any areas that may need improvement and commend good practices that are already in place. Keep the advice actionable and concise."* We will include the full graph JSON as context (or a summarized form of it) so the LLM can base its advice on the user's specifics.

Input: The confirmed graph data. If the graph is complex, we might summarize it or flatten it into key facts for input. But since we only have ~10 Q&A, it should be a manageable size to include as JSON or as a list of bullet points in the prompt. We must be mindful of token limits of the API, but a 10-node graph is small. The prompt might say: *"The user's GDPR profile is as follows: - Industry: Healthcare; - Has DPO: Yes; - … (etc). Given this, write a response… "*.

Output: A well-formulated **paragraph or bullet list** of recommendations. The style should be encouraging and clear. For example, it may produce: *"It's good that you have a Data Protection Officer; this ensures accountability. Given that you handle healthcare data, I recommend implementing encryption for personal data at rest if you haven't already. You mentioned using Amazon S3 – ensure that access to those buckets is restricted and monitored. Also, consider establishing a formal process for Data Subject Access Requests (DSARs) since those are critical under GDPR. …"* and so on. Essentially, it should cover any missing best practices or compliance measures relative to GDPR. The guidance should not scold but rather guide (we will emphasize positivity and helpfulness in the prompt). If possible, the LLM could also reference relevant GDPR articles or standards in the advice, but brevity is important for MVP.

We will also provide a **"system" message for the LLM API** in each call as described, and the user input (when applicable). The OpenAI Chat Completion API allows a system message, user message, etc. In our usage: - For question generation and guidance generation, the user message might actually not be from the user but from our app (containing the graph data and instructions), and the system message defines the role. - For graph update, similarly, the user message will contain the graph+answer.

Each of these prompts will be carefully tested in isolation with the LLM to fine-tune the instructions (this may require iterative prompt engineering in practice). The LLM should be *coaxed to only output the required content* (especially for the JSON). Techniques like asking it to output only JSON, perhaps framing

it as a function output, can help. The new OpenAI function calling feature could enforce a JSON schema, but for simplicity we assume just prompt-based formatting.

Importantly, all interactions with the LLM should include a directive to remain within role and not produce irrelevant text. We'll also include instructions for the LLM to maintain a **friendly tone** whenever it's producing user-facing text (questions or guidance). This means phrasing questions in a polite, conversational way and keeping the final advice constructive. The user should feel they are in a dialogue with a helpful assistant, not an interrogator. This approach is in line with making the experience engaging rather than tedious [9] [10].

For example, the system prompt for question generation might also say: *"The questions should be concise and clear. If possible, make them engaging (e.g., instead of a dry 'List your security measures', say 'Could you tell me about...'). Maintain a tone as if a colleague is asking, not a form."* Similarly, the final guidance prompt will ensure the response sounds like personalized advice, not a generic printout.

Front-End Implementation Details

This MVP will be implemented as a purely **client-side web application** (e.g., an HTML/JavaScript single-page app). The components and their responsibilities are outlined below:

- **Chat Interface:** A main area will display the conversation between the system and the user. The system's questions and final advice will appear in chat bubbles (or simply as paragraphs), and the user will have a way to input answers (a text input box for open-ended answers, and possibly quick Yes/No buttons for the confirmation step). Each time the system (LLM) outputs a question or the final guidance, it should be appended to the chat log. We will prepend a brief greeting or explanation at start (for example: *"Hi, I'm here to help assess your GDPR practices. I'll ask you a few questions to understand your current setup, and then provide some guidance. Let's get started!"*). This sets a friendly context. Then the first LLM-generated question is shown.

- **User Input Handling:** When the user submits an answer, the app will disable input and show a "loading" indicator (like "..") while the LLM processes. This is where we call the LLM API for the graph update. Once the JSON is returned and processed, we immediately call the LLM again for the next question. These two calls happen back-to-back without user intervention. The user just sees that after they answered, the system "thinks" and then responds with the next question. Using streaming for the question generation call can make the UI feel very responsive – the question starts appearing as soon as it's generated. This aligns with a modern chatbot UX.

- **Local Storage and API Key:** We will require the user to supply their OpenAI API key (assuming use of OpenAI's GPT-4 or similar). We'll provide a field in the UI (for example, a modal or a settings section) where they can paste their API key. The app will store this in `window.localStorage` (so it persists for that browser). All fetch calls to OpenAI endpoints will pull this key for authorization. No keys or data are sent to any server of ours. We should caution in the UI about not sharing sensitive personal data, since their answers are going to an AI API (for GDPR compliance questions this should be fine, it's more about organizational practices, but we should still make them aware of data usage).

- **State Management:** We will keep the current graph in a JavaScript variable (and optionally sync it to localStorage as well, in case of refresh or debug). After each LLM graph update, we overwrite the local graph state. We also keep track of how many questions have been asked so far (a simple counter) to decide when to stop. We keep the list of remaining topics (if using that

method) either in the prompt or in our code. Alternatively, we could rely on the LLM to signal when done, but counting to 10 is straightforward. We also track when we are in the confirmation phase or finished, to manage the UI accordingly (e.g., don't allow free text input during confirmation, only Yes/No buttons).

- **Visualization of Graph (Optional):** It would be a nice touch to visualize the knowledge graph being built. For MVP, this can be quite simple – even a text-based display of nodes and relationships might suffice (like a tree or outline). If time permits and using a front-end library is allowed in vibe coding, we could incorporate a small graph visualization library (such as **D3.js**, **Vis.js**, or **Cytoscape.js**) to render nodes and edges. For example, nodes could be shown as bubbles with labels ("Healthcare", "DPO: Alice", etc.) connected to the central "User" node. This visual can update each time the graph grows, reinforcing to the user what the system has learned [11] [12]. However, implementing a full dynamic visualization might be complex to prompt-code reliably. As an alternative, the **confirmation summary text** is a form of visualization – it lists the graph's content in human-readable form, which we are already doing. We can consider the graphical view an enhancement for later. The key is that the user should *feel the graph building up* in some way, rather than the Q&A being a black box. Even just showing the bullet list of facts at the end achieves that (and during the conversation, the user implicitly knows what they've told the system).

- **Styling and UX:** We will keep the UI clean and simple. A basic CSS style can make the chat messages distinct (perhaps the system's messages in one color bubble and the user's in another). The vibe should be professional yet approachable (in line with the friendly tone of the LLM's questions). We should ensure the interface is responsive (works on desktop or tablet; mobile if possible). Since this is an MVP and "vibe coded," we won't aim for pixel-perfect design but rather functional clarity. We should also handle basic errors: e.g., if the LLM returns invalid JSON (we can catch a JSON parse error and maybe retry or show an error message), or if the API call fails (network issue or invalid API key – in which case we prompt the user to check their key or connection).

- **No Server Storage:** All data stays in the browser. If we want to allow the user to come back to their session, we could store the graph in localStorage at each step. That way, if they refresh, we could (in theory) re-load the graph and perhaps even the conversation. But conversation replay would require storing all past messages too. To keep scope limited, we might not implement resume functionality now. Each session is ephemeral (the user can always restart the Q&A). Ensuring no server involvement simplifies compliance as well, since we're dealing with potentially sensitive compliance info – it lives only in the user's browser and in the LLM's processing.

Example Scenario (Illustrative)

To clarify how all the pieces come together, consider an example run-through with a hypothetical user:

- **Startup:** The user opens the web app. They paste their API key when prompted. The interface greets them and starts the interview.

- **Q1:** *System (LLM):* "Hi! To start with, what industry is your company in?" (This was either predefined or generated. The system prompt ensured it's friendly and concise.)
  User: "We're in the healthcare sector."
  Graph Update: LLM gets graph `{"nodes":[{"id":"persona","type":"Organization","name":"UserOrg"}]}` and answer

`"We're in the healthcare sector."` and returns updated graph JSON. The JSON now includes something like a node for healthcare industry and an edge linking it: e.g. `"nodes": [..., {"id":"industry_healthcare","type":"Industry","name":"Healthcare"}], "edges":[..., {"source":"persona","target":"industry_healthcare","relation":"industry"}]`.
Next Question: LLM is then given the new graph and the list of remaining topics. It knows industry is handled, so it asks something else, maybe *"How many employees do you have, and do you operate only in the EU or globally?"* (It combined two related sub-questions perhaps about company size and data jurisdiction – which is fine, the user's answer can cover both). The UI displays this.

- **Q2:** *User:* "We have 250 employees, and we operate across Europe and Asia."
  Graph Update: The LLM adds a node or attribute for "employee count:250" and perhaps a node "Operating regions: Europe, Asia" (or separate nodes for each region with relation "operates_in"). These would be connected to the persona node.
  Next Question: The LLM sees that company size and regions are captured, moves to the next target. Possibly it asks: *"Do you have a Data Protection Officer (DPO) appointed?"*.

- **Q3:** *User:* "Yes, our Chief Privacy Officer also acts as the DPO."
  Graph Update: The LLM might add a node for "Data Protection Officer" with a relation indicating it exists. It might even add detail that the role is filled by the Chief Privacy Officer (if we choose to capture that detail, though it might be unnecessary for the main graph beyond just noting existence of a DPO).
  Next Question: Perhaps: *"Do you maintain records of processing activities (ROPA) as required by GDPR?"*. (This dives into a compliance detail; if the LLM knows the term it might use it, or it might phrase it in lay terms like "documents of what personal data you process").

- **...** and so on for up to 10 questions, covering things like data subject request handling, data breach response plan, data retention policy, etc. Each time, the graph grows. The adaptive nature means if, say, the user in one answer mentions something unexpected ("We don't have a formal retention policy, but we do have an encryption system in place"), the LLM could pick up on "encryption system" and later ask a follow-up about it or at least add it to the graph. The question flow is not strictly linear – the LLM will intelligently skip topics already answered and dig deeper on those that are relevant [3] (for MVP we mostly rely on user's answers and our static list of topics, but the LLM could be smart enough to adjust order).

- **After Q10:** The system decides enough questions have been asked. It transitions to **confirmation**. The UI might say: *"Thanks for answering these questions. Here's what I've learned about your GDPR setup – please confirm if this is correct:"* and then list the facts:

- *Industry:* Healthcare
- *Employees:* 250 (operating in Europe and Asia)
- *DPO:* Yes (Chief Privacy Officer serves as DPO)
- *Records of Processing:* (user's answer, say they answered "no formal ROPA")
- *Breach Response Plan:* e.g. "Has procedure in place for data breach notifications" or not, etc.
  (Each corresponding to how they answered each question.)

User: clicks **Yes, correct** (assuming it's all accurate). If they clicked No, ideally we'd allow editing some entries or would loop back, but as noted we won't implement that in this iteration.

- **Guidance:** Now the app calls the LLM with the final graph and the guidance prompt. The LLM returns, for example:

"It looks like your organization has a solid foundation: having a DPO and conducting employee privacy training are excellent. One area to improve is documenting your processing activities – GDPR requires maintaining records, so setting up a ROPA should be a priority. Also, since you operate in Asia as well, ensure you are aware of any data transfer requirements (e.g., Standard Contractual Clauses for transfers outside the EU). I recommend establishing a formal data retention policy because you mentioned not having one; this will help you avoid holding data longer than necessary. Overall, you're on the right track – focus on those documentation aspects and you will significantly strengthen your GDPR compliance."

The UI displays this as the final answer from the assistant. Possibly we style it a bit differently or label it "Guidance Report" for clarity.

- **End:** The user can scroll through the whole Q&A and see how their answers led to the recommendations. The experience should feel like an interactive consultation. And because we structured the data under the hood, the advice was tailored and the user also got to see a summary of their inputs, enhancing trust in the output [7].

This scenario demonstrates how the pieces fit. Of course, during development we will test various paths and refine the prompts to ensure the LLM asks relevant questions and cleanly updates the graph.

Technical Considerations and Next Steps

For the LLM, we'll likely use GPT-4 (or GPT-3.5 if cost is a concern, though GPT-4's better understanding may be needed for reliable graph outputs). Each question/answer cycle involves two LLM calls, so with 10 questions we have about 20 calls, plus one for guidance (21 total). We should optimize prompts to stay within reasonable token counts. The graph JSON will grow but should remain small (tens of lines). Using a compact schema and possibly stripping irrelevant parts (like we don't need the entire history, just the current state) will help. If using OpenAI's streaming, we need to handle the streaming response for the question text; for JSON outputs we likely wait until the full completion (since JSON must be complete).

The design is intentionally **serverless** and uses modern web development simplicity: essentially HTML, minimal CSS, and JavaScript for API calls. If writing this via vibe-coding (AI-generated code), we'd instruct the LLM (the coder) to perhaps use fetch for API calls, handle asynchronous flows (promises or async/await), and update the DOM accordingly. No external libraries are strictly required, though something like a markdown renderer could format the final guidance nicely if needed. However, the simpler the better for now.

System prompts for each phase recap: We will prepare the following prompt templates for the LLM: - *SystemPrompt_Q*: Role = Question Asker. *"You are an AI that asks the user questions to gather info about GDPR compliance. You have these goals... (list topics) ... and the current knowledge graph. Ask one next question. Friendly tone."* - *SystemPrompt_Update*: Role = Graph Updater. *"You are an AI that updates a JSON graph. Only output JSON. Here's the current graph and new answer. Follow schema X."* - *SystemPrompt_Guidance*: Role = Advisor. *"You are an AI GDPR consultant giving advice based on the profile. Provide recommendations in friendly tone."*

Each will be accompanied by the necessary data (graph/answer) as user messages. We'll also ensure to include in these prompts any specific format requirements (like JSON only, etc.).

Finally, although this MVP is focused only on GDPR Q&A, the architecture is generic. In the future, we could extend the graph to a fuller "persona graph" that includes not just GDPR but other domains (security practices, business context, etc.), and incorporate external data (for instance, pulling some public info on the company to pre-populate the graph) [13] [14]. We could also integrate a proper graph database (like the mentioned MGraph-DB in Dinis's research) to persist and query these graphs, and use serverless functions to offload the LLM calls. But those are beyond the scope of the current task. The immediate next step after this MVP works is likely to refine the prompts (making the conversation as efficient as possible) and improve the UI (possibly adding that graph visualization or more interactivity in the confirmation step).

In conclusion, this technical brief provides the blueprint for an LLM-powered, no-backend chatbot that **dynamically builds a semantic graph of a user's GDPR compliance posture and provides tailored recommendations**. By following this design, an LLM (with "vibe coding" techniques) should be able to generate the necessary front-end code and logic. The result will demonstrate a powerful pattern: using GenAI to turn a boring compliance questionnaire into an engaging, interactive experience that yields a personalized knowledge graph and actionable insights [12] [7]. This not only makes data collection more pleasant for the user, but also enhances the quality and trustworthiness of the output (since the user is involved at each step) [15]. The MVP will serve as a foundation for further development, such as expanding the graph's scope (persona graphs) and integrating deeper analytics or multi-session capabilities in the future.

---

[1] [2] [3] [4] [5] [6] [7] [9] [10] [11] [12] [13] [14] [15] User-Driven Semantic Persona Graphs Powered by GenAI - Dinis Cruz - Research Hub
https://docs.diniscruz.ai/2025/06/14/user-driven-semantic-persona-graphs-powered-by-genai.html?trk=public_post_comment-text

[8] HowTo: Generating Terminology Knowledge Graphs using ChatGPT - Tips, Tricks, and HowTos - OpenLink Software Community
https://community.openlinksw.com/t/howto-generating-terminology-knowledge-graphs-using-chatgpt/3914