Reflection Q&A

1. What are the pain points in using LLMs?

LLMs often required repeated clarification. Even after rewording prompts several times, the output could still drift away from what we wanted. Creativity was another weak spot: instead of offering novel ideas, the models often gave us uniform or repetitive answers. Context retention was also an issue—long conversations caused the model to "forget" earlier details and sometimes contradict itself. When we used both Gemini and ChatGPT, we saw this inconsistency play out even more clearly: they often gave completely different takes on the same requirement, and sometimes the same model shifted its answer when we re-asked. We also ran into hallucination, where features appeared that sounded convincing but weren't in scope for WolfCafe or didn't line up with regulations. Amplification usually worked well, but condensation was messy—rather than prioritizing, the models just repeated or rephrased existing points. Some of this came from the models themselves, but a lot came from us as the prompters. Whenever we crammed too much into one long prompt, the model lost track and contradicted itself, and when we left prompts too open-ended, we just got generic, low-creativity results. Another recurring gap was traceability: the outputs didn't connect back to stakeholder needs or regulatory sources, so we had to add that layer ourselves. In the end, we realized that the quality of results reflected both the limits of the LLMs and the "pot of the prompter"—our prompting style shaped the outcome just as much as the models did.

2. Any surprises? Eg different conclusions from LLMs?

One surprise was how differently Gemini and ChatGPT framed the same problem, even though we gave them the same instructions. Gemini pushed for a very minimal MVP, cutting aggressively and suggesting we focus only on the end-to-end order loop (place → pay → fulfill → pickup), while ChatGPT kept more of the "extended" features like order confirmation, SSO login, dietary filters, and quick reorders. In some cases, the models even labeled the same feature differently—Gemini called out "Admin Manages Sales Tax," while ChatGPT referred to it as "Admin Updates Sales Tax Rate," which shows how wording shifted even though the intent was the same. Another surprise was how polished the use cases looked once we forced them into a structured template (Preconditions, Main Flow, Subflows, Alternatives). Both models produced outputs that felt ready to paste into our 1c1 deliverable, but when we compared them side by side, the priorities diverged: Gemini emphasized simplicity and feasibility, ChatGPT leaned toward coverage and stakeholder satisfaction. This reinforced the idea that LLMs act like different "stakeholders" with their own biases, and that part of our job was reconciling those perspectives into a single MVP scope.

3. What worked best?

What worked best was using structured prompts and templates. When we framed use cases in the format of Preconditions, Main Flow, Subflows, and Alternatives, both Gemini and ChatGPT produced clear, well-organized outputs that were almost ready to drop into our deliverables. Splitting the

workflow into amplification (generate many ideas) and condensation (narrow to MVP) also worked well, with Gemini excelling at trimming and ChatGPT at preserving stakeholder needs. Comparing outputs across the two models surfaced trade-offs we might have missed. We also had success building a small RAG pipeline: extracting text from assigned papers and articles with Python, saving them as .txt, and feeding them into the LLM. Grounding prompts in these curated sources noticeably improved relevance and alignment with expectations.

3. What worked worst?

What worked worst was leaving prompts too open-ended. When we simply asked an LLM to "generate requirements," the results were generic, repetitive, or off-scope. Condensation was another weak spot — instead of true prioritization, the models often just rephrased or duplicated requirements. Long, overloaded prompts also backfired, with the model losing context and contradicting earlier details. Finally, while the outputs often looked polished, they sometimes included hallucinated features or compliance claims that weren't realistic, which created extra review work.

4. What pre-post processing was useful for structuring the import prompts, then summarizing the output?

For pre-processing, the main step was extracting text from the professor's assigned research papers and regulatory articles. We used Python to scrape or download the content from links, convert it into text files, and then fed that into the LLM. This gave the model grounded material that was directly tied to the course readings. We also structured our prompts in use case format (Preconditions, Main Flow, Subflows, Alternatives), which improved clarity and consistency. For post-processing, the most useful step was reconciling the different outputs from Gemini and ChatGPT — trimming repetition, removing hallucinated features, and deciding which use cases to keep or drop for the MVP. This filtering and comparison work was necessary to get a clean, coherent set of requirements we could actually use in our deliverables.

5. Did you find any best/worst prompting strategies?

On the pre-processing side, we learned that *chunking inputs* was often better than dumping full articles. Feeding the LLM shorter, focused excerpts gave us more precise answers tied to a specific regulation or theme. On the post-processing side, the most useful step was keeping an explicit audit trail of where requirements came from. By noting "this came from Gemini" vs. "this came from ChatGPT" vs. "this came from Article X," we could explain our decisions more clearly in the reflection. This also made it easier to justify why certain features were kept or dropped, since we could point back to both stakeholder needs and model tendencies.