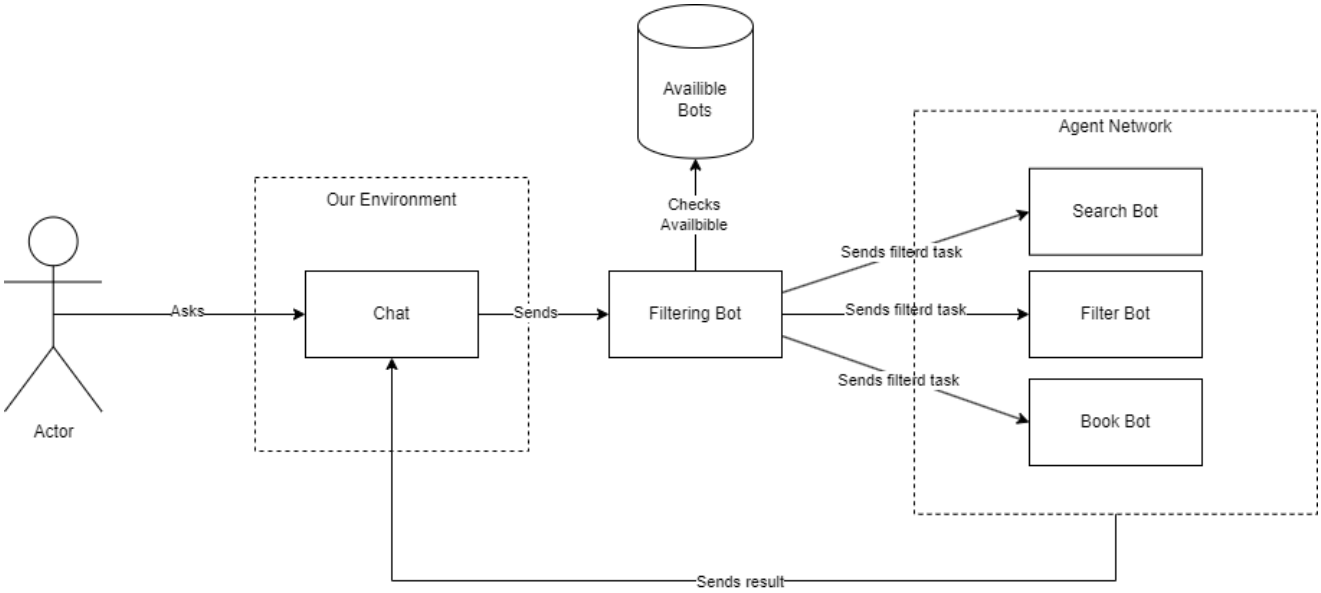
Architectural Designs

In this file I will put all the architectural designs that we made as a group for the project called 'Collaborating Bots'. For each design, there will be a small description on why this design was created or thought of.

Architecture Diagram 1

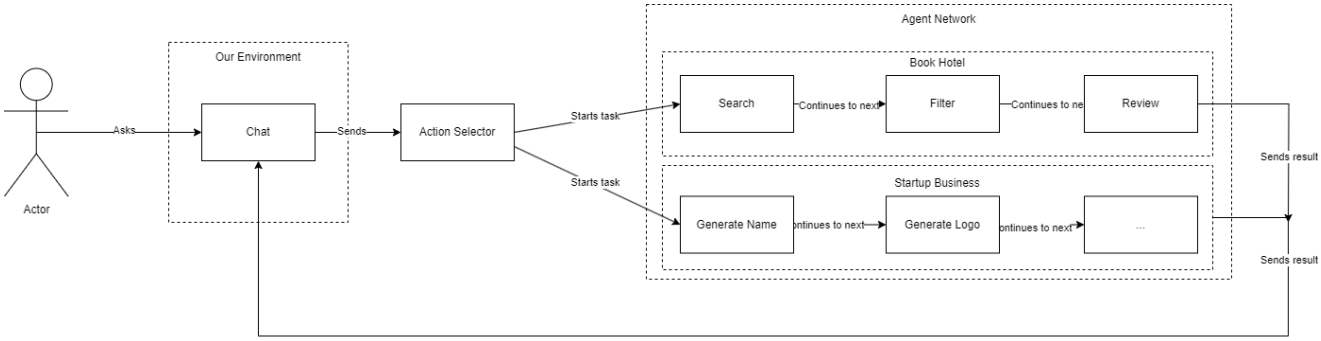
The first architecture that we made was a very basic one, it did not yet include any of the mentioned topic like Activity Pub etc. At this point, we already decided that there would be some kind of UI (chat interface) where a user will input a prompt that will be handled by one or more agents. To know what the user's prompt means, there would be some kind of filtering going on. This filter is hooked up to a database that knows about the available agents, it will then send a request to those agents who will process the prompt and send it back to the UI.



Architecture Diagram 2

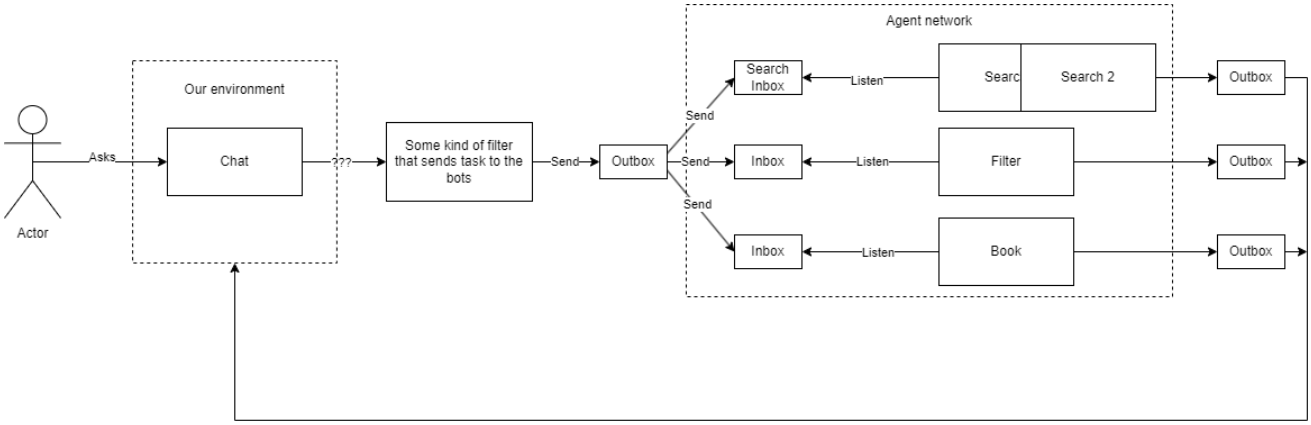
As we wanted to remove the database rom the architecture as it would be some kind of manager class, we decided to remove it (for now) from the diagram. This architecture was more of a set route for a user's prompt. For example, I as a user want to book a hotel. The system will then go into the Book Hotel route, where it has pre-defined steps to take before

sending the result back. As this was not a very scalable architecture, we decided to steer away from this approach.



Architecture Diagram 3

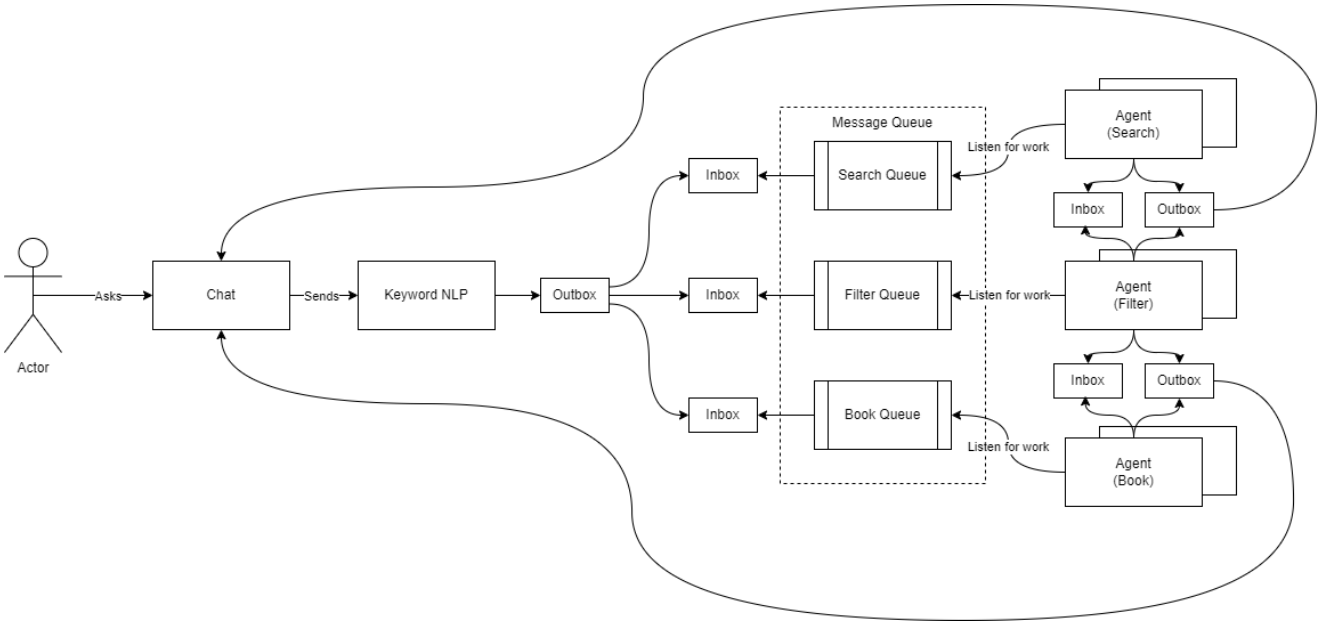
The third architecture focuses heavily on how to achieve the group project using the principles of the Activity Pub protocol. Once the prompt goes through the filter, it will then send to its own outbox with a JSON object wrapper in an activity so that the ActivityPub server will send it to the corresponding inboxes. An inbox could also be a shared one where multiple agents are hooked up to. Once the agents finish the processing, it will be sent back to the UI.



Architecture Diagram 4

After our last design, we started to look into some more concepts to make the system more scalable. Reno started to look into some RabbitMQ and started to think about how this could be implemented together with the ActivityPub protocol and came up the following design. In this design, the key parts are about the same as they were before. So, it goes from user prompt, filter, ActivityPub outbox and inbox. After this step, he added a message queue, one for each topic that was available. The agents then

could subscribe to this queue to find available work, this would also solve the problem of distributing work over agents of the same category.

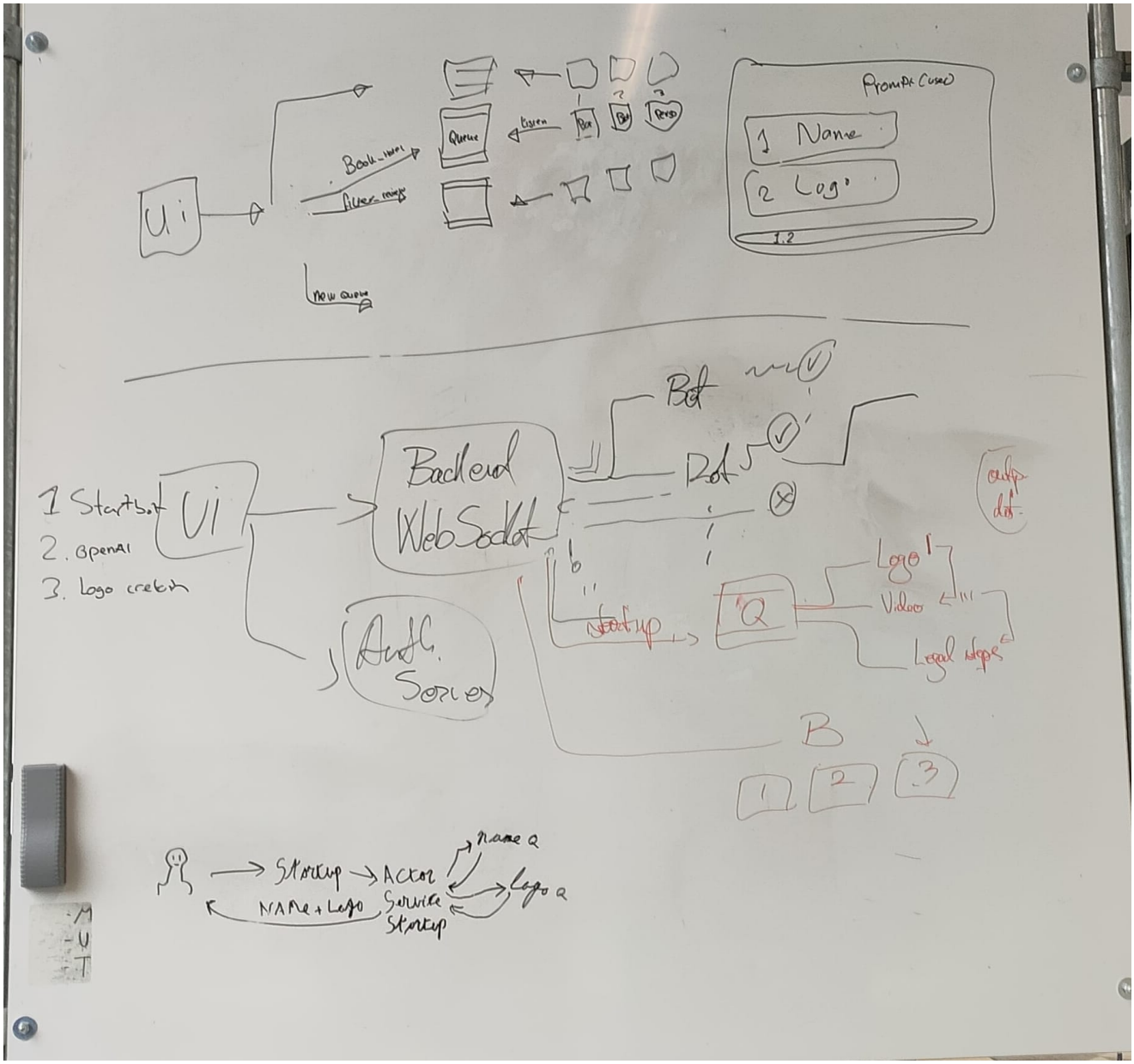


Architecture Diagram 5

At this point we decided to forget the ActivityPub for now as this concept was not yet clear to us on how to implement it into our system. Iulia came up with an architecture where the UI would be connected to a backend server via WebSocket's. In this service, it would filter the request and check to what agents it needed to send. These agents would also be connected to the same WebSocket. Doing it like this would create a fast real-time system to will send data back and forward over the WebSocket's.

After Iulia presented here idea, Reno also pretended his own idea about this. Which included the queues from previous answers. This would make sure that the bots could be subscribed to a certain topic queue and that would not be any database needed as the queue name was what the bot

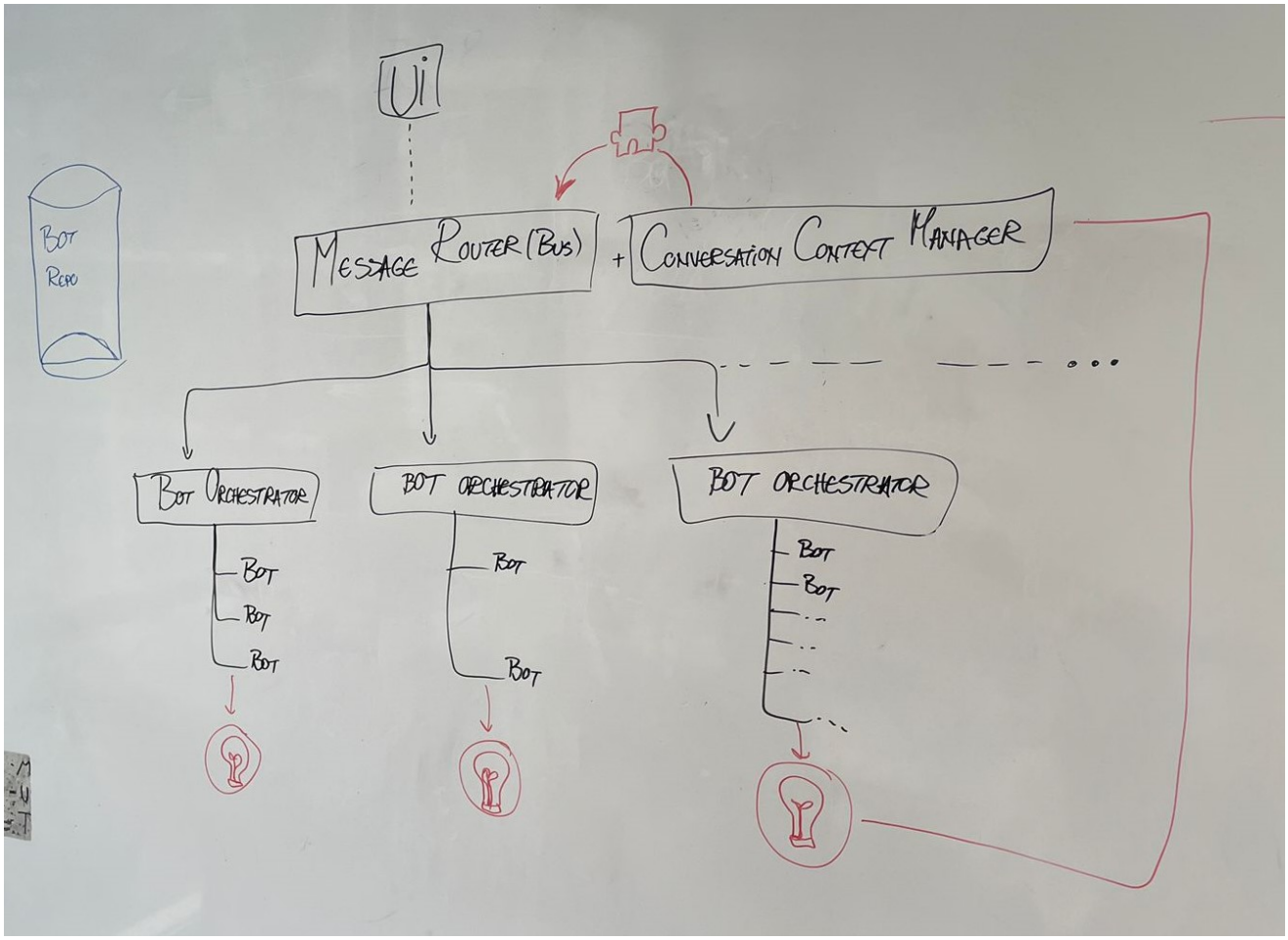
does.



Architecture Diagram 6

After some time, Iulia came up with the following diagram. At this point we already did some more research about message queues and message bus and ultimately decided to go with a message bus as this was a more scalable approach. We added the database again but did not yet connect it to anything as we are still not sure to what it needs to connect. And a 'Conversation Context Manager' was added, what was actually a fancy word for filter. The flow op this application is as followed. UI send prompt to message bus which filters it and sends it to bot orchestrator which would orchestrate the agents. Once there is a result it will be sent back to

the UI.



Architecture Diagram 7

To clarify the previous architecture a bit more, Reno drew how it actually would work without the fancy words. So, UI sends prompt over WebSocket's to the backend server, which will send the request to the filter. This filter will check what kind of topic that the prompt is (e.g., travel, learning, etc). This would then be sent to a topic in the message bus where the bot orchestrators would be subscribed to. The orchestrators would then send the tasks to the agents that need to do a task to achieve the outcome, and the orchestrators will send it back to a response topic in the message bus. The backend of the UI is subscribed to this topic and

will send the result back to the UI over WebSocket's.