Research project specification

1. Introduction

Estimated reading time: 20 minutes.

This document is a specification of the application proposed in the research project “Helping users navigate data specification”. The project aims to enhance the Dataspecer tool with a standalone application which will allow users to ask questions about data that they have a data specification of.

Dataspecer is a tool designed for managing and modeling data specifications. It helps users create structured data schemas from ontologies and export them into various formats such as JSON, XML, and CSV. The Dataspecer manager lets users create and manage Dataspecer packages – these packages will be used as data specifications in this project.

This specification describes the interaction between the user and the application, the conversation model between the user and the application, user stories and use cases. It presents the architecture of the proposed application and lists the technologies that will be used to build the application.

2. Motivation

In an organization, there are usually only a few people who fully grasp the whole domain ontology. An average person might be interested in some data that is available in the organization’s database but does not know how to construct a query to get the data they want.

My proposed application will not only help users construct the query they need to get the data they want but also offer them items from their organization’s data specification which might be of interest to them. Users can decide to add the offered items into the query that the application is constructing for them thus expanding the query. The reason why the application would offer these items for query expansion is because users often do not realize that their data specification contains those things and that they could have asked for them in their initial question. By offering these items, the application is helping users navigate the organization’s data specification as stated in the project’s name.

3. How the application achieves its goal

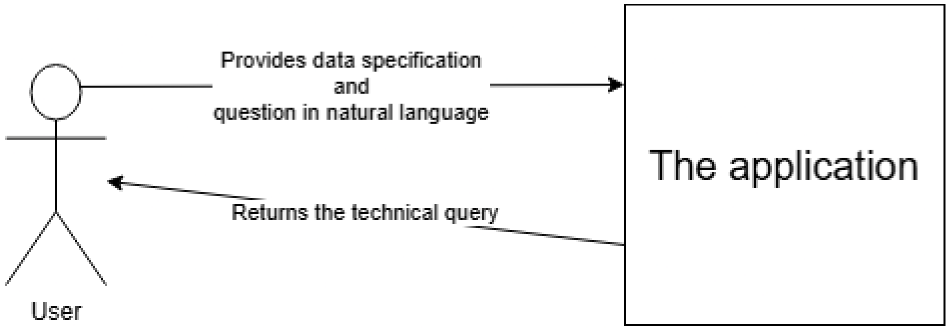
This chapter describes the input and output of the application from the user's point of view, the user-application interaction and an overview of how the application generates output for the user.

3.1. Input and output

From the perspective of the user, the output of the application is a query that can be used to get the data they want. The user gives the application a description of their data specification and their question in natural language. The application will help the user expand the query it is constructing from the initial question, but the end result is always a query that can be run (*see Figure 1: The application's input and output*). The query language that I have chosen as the output of this application is Sparql, but it could be some other language like for example SQL.Figure 1: The application's input and output

In this project, the data specification provided by the user will always be a Dataspecer package. The Dataspecer manager tool will let the user choose a package and upload it to my application. The Dataspecer manager will then redirect the user to my application where they can ask their question.

Figure 1: The application's input and output


Provides data specification
and
question in natural language
Returns the technical query
User
The application

3.2. User-application interaction

The user interacts with the application in a manner that resembles a chat bot similar to ChatGPT, Gemini etc. but there are key differences. The user interface will be a regular chatting interface that most people are familiar with. It will feature a chat box where users can write their messages. Both user messages and answers from the application will be displayed above the chat box.

When the user provides the data specification and writes their question, the application will reply with an answer that contains the constructed Sparql query and a list of data specification items. This list contains the items related to the user's question. The user will be able to click on each item to see a short summary of that item. While viewing the summary for related items, the user will have the option to add the item to their question and construct a new Sparql query with the item included.

Whenever the user chooses to add a related item to expand the query, the application will generate and display a question preview in the chat box. This question preview is the user's original question in natural language but with the newly added item from the data specification. From this point onwards, I will refer to this preview as a **suggested message**. After selecting the items that they want to add to the query, the user can directly send the suggested message to generate a new Sparql query, or the user can first modify the suggestion and then send the modified version. The application will once again process the user's question and send an answer. The user can now select more items to expand the constructed Sparql query.

The interaction should continue for as long as there are items to be offered to the user or until the user stops choosing items to add to the query. At this point I want to emphasize that I have described two possible ways to work with the application. The first one is that the user can manually type out every message just like any regular conversation. The second option is that the user only types the first message and after that they click on items in the answer, choose whatever they want to add to the question and send the question with or without modifying the preview. Finally, there is also the option to combine both approaches.

3.3. Constructing a query from the user's question

When the user writes their question in natural language, we will try to identify and map keywords from the user's question to items in their data specification. This mapping will give us a subset of the data specification. As an example, imagine a data specification about the sport badminton. It can contain tournaments and attributes of the class tournament, it can contain badminton players and their attributes, it can also contain companies that play roles in the sport (*see Figure 2: Class diagram of the badminton specification*). For example, the user could ask the question "Show me tournaments which have taken place this year and have participants from the Czech Republic." We can map the word "tournaments" to the class "tournament" in the data specification, we can map the words "taken place this year" to the attribute "date held", we can map "participants" to the relationship between "tournament" and "player", we can map "from the Czech Republic" to the player's "country". This mapping gives us a subset from the potentially very large data specification that is badminton. From now on, I will refer to this subset as a **data specification substructure**.

Figure 2: Class diagram of the badminton specification

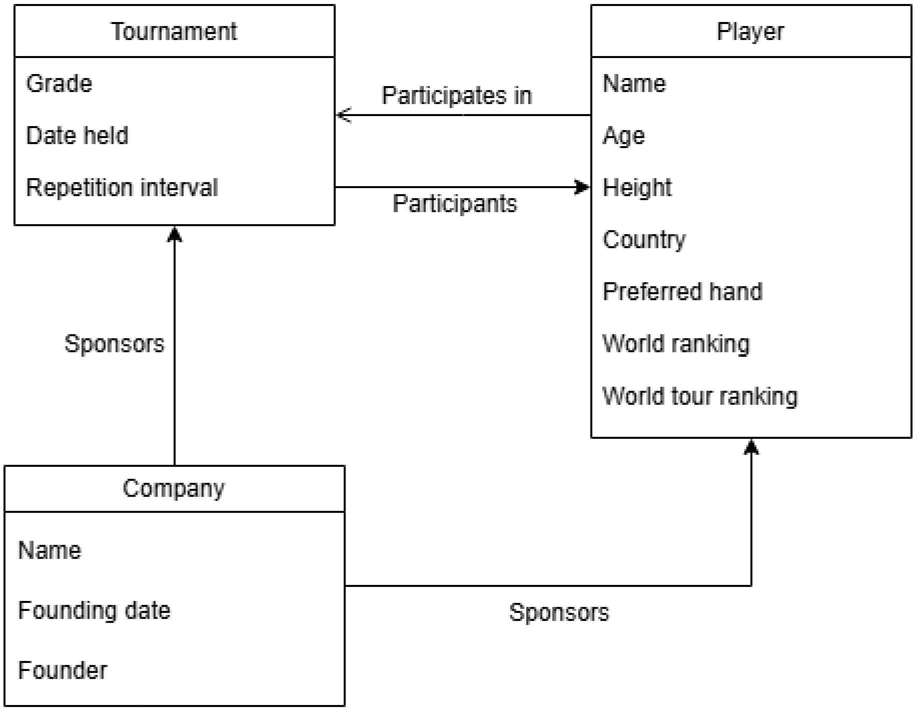
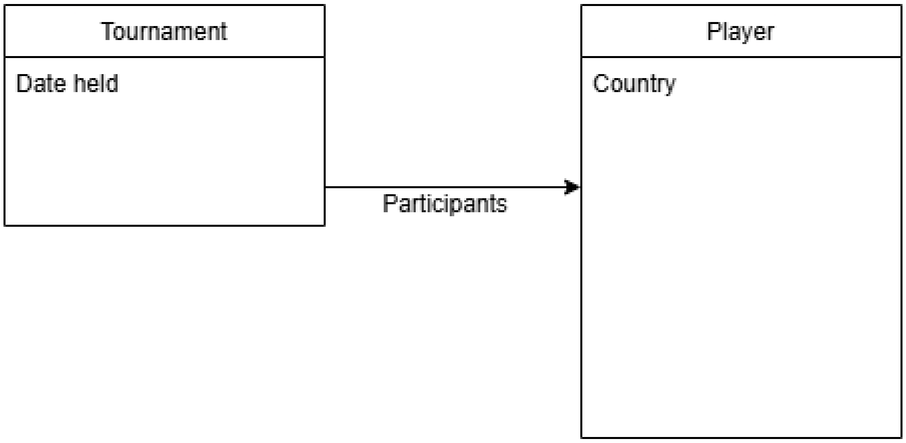


Figure 3: Data specification substructure



From the "perspective" of the application, the output is the data specification substructure described above. We will use this substructure to generate the Sparql query and to generate questions previews for the user.

If we fail to map any keywords from the user's question to any items in the data specification, then the data specification substructure is an empty set. In this case, the application will respond to the user with a message saying that it did not find any suitable matches from the data specification and cannot help them construct a query.

3.4. Offering items related to the question

To help the user navigate the provided data specification, the application will offer items from the data specification which are relevant to the user's question. After we successfully map the user's question to the items in the data specification and get a substructure, we will use a large language model (further abbreviated as LLM) to ask for items that are relevant to the user's question. These could be other attributes of the already mapped classes, it could be other classes in the vicinity of the mapped classes, or it could be something else. What items should be offered is subject to experimentations with the LLM and at this point it is hard to define precisely.

Whenever the user chooses an item to add to the query, we will add that item to the substructure that we are building so that we can use it to generate a new Sparql query for the user.

3.5. Generating suggested messages

When the user chooses an item to add to the question, we take the current substructure and add the user's chosen item to it. Then with the help of the LLM we will use this substructure to generate the suggested message for the user.

As an example, in our previous example with the badminton data specification, the user's question was "Show me tournaments which have taken place this year and have participants from the Czech Republic." Suppose the user chooses the item "tournament sponsors" to add to their question. Then the suggested message could be: "Show me tournaments which have taken place this year and have participants from the Czech Republic and also the sponsors for those tournaments."

3.6. The usage of large language models

We will use an LLM for the following tasks described in previous sections.

1. Map the user's question to items in the data specification and get the data specification substructure.
2. Get items from the data specification which are relevant to the user's question. The application will offer these items to the user to expand the question.
3. Generating suggested messages.

This application must be independent of any specific LLMs or LLM frameworks or prompting strategies. Prompt templates will be sufficiently separated from the code to ensure easy configuration. For the purposes of development and testing, we will use the LLMs deployed using the Ollama framework on the KSI server.

4. Functional requirements

This chapter describes the application via user stories and use cases.

4.1. User stories

1. The user can choose a Dataspecer package that they want to use as a data specification in this application.
2. The user can ask questions about data. The data they ask about should conform to the chosen data specification.
3. The user will receive a Sparql query as an answer to their question. This Sparql query can be run to obtain the data that they asked for in their question.
4. The user can expand their questions with the help of the application.

4.2. Use cases

I have included mockups of the UI in this section to help the reader better understand the use cases.

4.2.1. Choosing a data specification

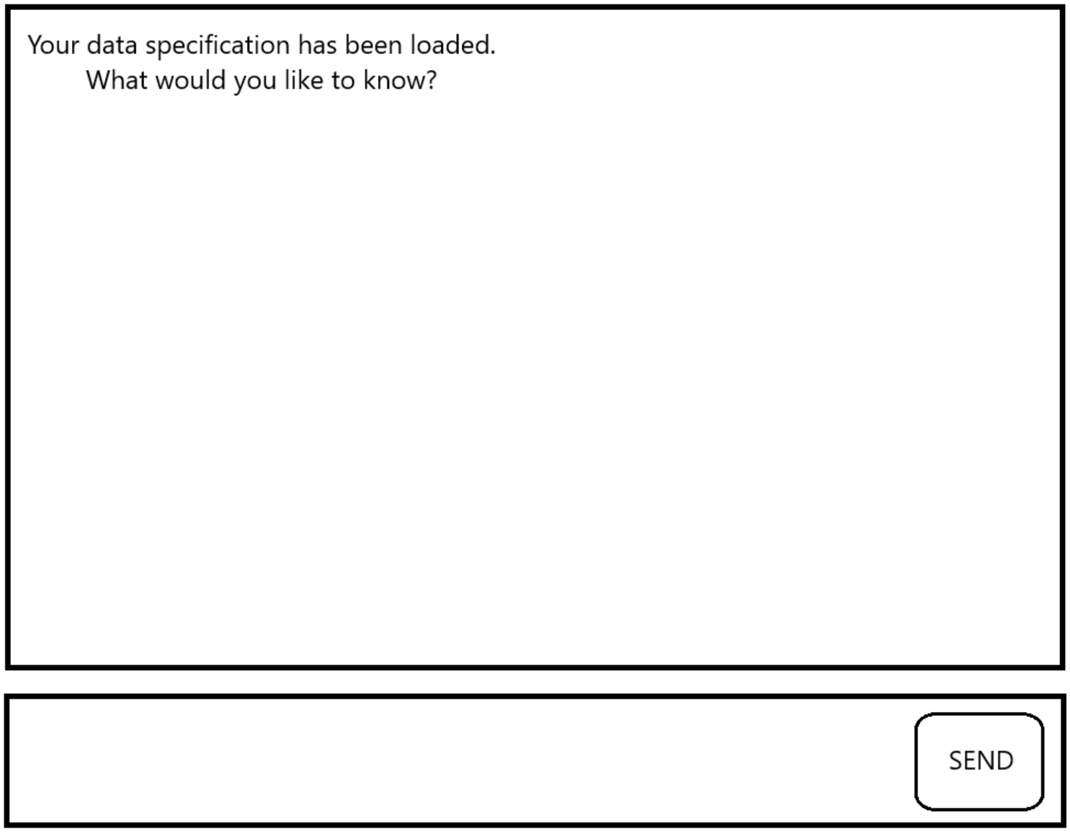
Precondition: The user can access and use the Dataspecer manager tool.

1. The user navigates to the Dataspecer manager tool.
2. The user identifies the Dataspecer package that they want to use as a data specification.
3. The user clicks on a button in the drop-down menu next to the Dataspecer package and chooses the option to use that package for this application.
4. The Dataspecer manager redirects the user to the chatting interface of this application.
5. The application creates a new conversation for the chosen data specification.
6. The application sends the first message into the conversation prompting the user to ask their question.

Postcondition: The application has processed the Dataspecer package as a data specification and is ready to receive questions from the user.

The application will store all created conversations so that the user can come back to them in the future. The application will have a conversation management tab where the user can see all created conversations and delete or open them.

Figure 4: A new conversation



Your data specification has been loaded.
What would you like to know?

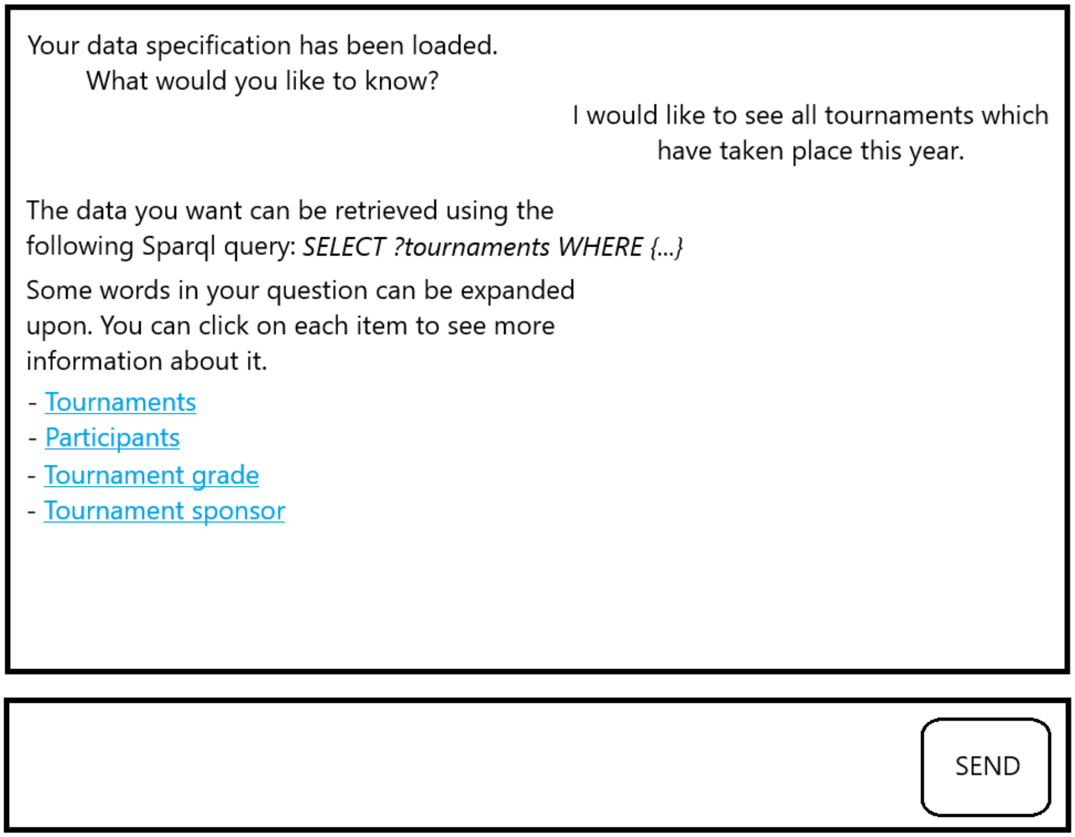
4.2.2. Sending a message

Precondition: The user has chosen a data specification and started a new conversation.

1. The user writes their question into the chat box and sends it into the conversation either by pressing enter or clicking on the SEND button next to the chat box.
2. The application responds with its answer to the question. The answer is added to the conversation as a chat message from the application. The response contains the following: (*also see Figure 5: An answer to the user's question*)
    a. A Sparql query constructed from the user’s question.
    b. A list of items from the data specification. The user can click on each item in this list to see a summary for that.

Postcondition: The user has received an answer with a Sparql query along with a list of items related to the question.

Figure 5: An answer to the user's question



Your data specification has been loaded.
What would you like to know?

I would like to see all tournaments which have taken place this year.

The data you want can be retrieved using the following Sparql query: *SELECT ?tournaments WHERE {...}*

Some words in your question can be expanded upon. You can click on each item to see more information about it.

- Tournaments
- Participants
- Tournament grade
- Tournament sponsor

4.2.3. Viewing item summaries

Precondition: The user has received an answer from the use case *Sending a message*.

1. Use clicks on an item in the list.
2. The application displays a window with a summary of the chosen item. If the item is not already in the original question, the user will see a button “Add item to my question”.

Postcondition: The user is able to view a summary for each item that the application has listed in its answer. The user has the option to choose items to add to their question.

For mockups of item summaries, see *Figure 6: Summary of item tournaments* and *Figure 7: Summary of item tournament sponsor*.

4.2.4. Expanding the current question

Precondition: The user can click on offered items and see their summaries.

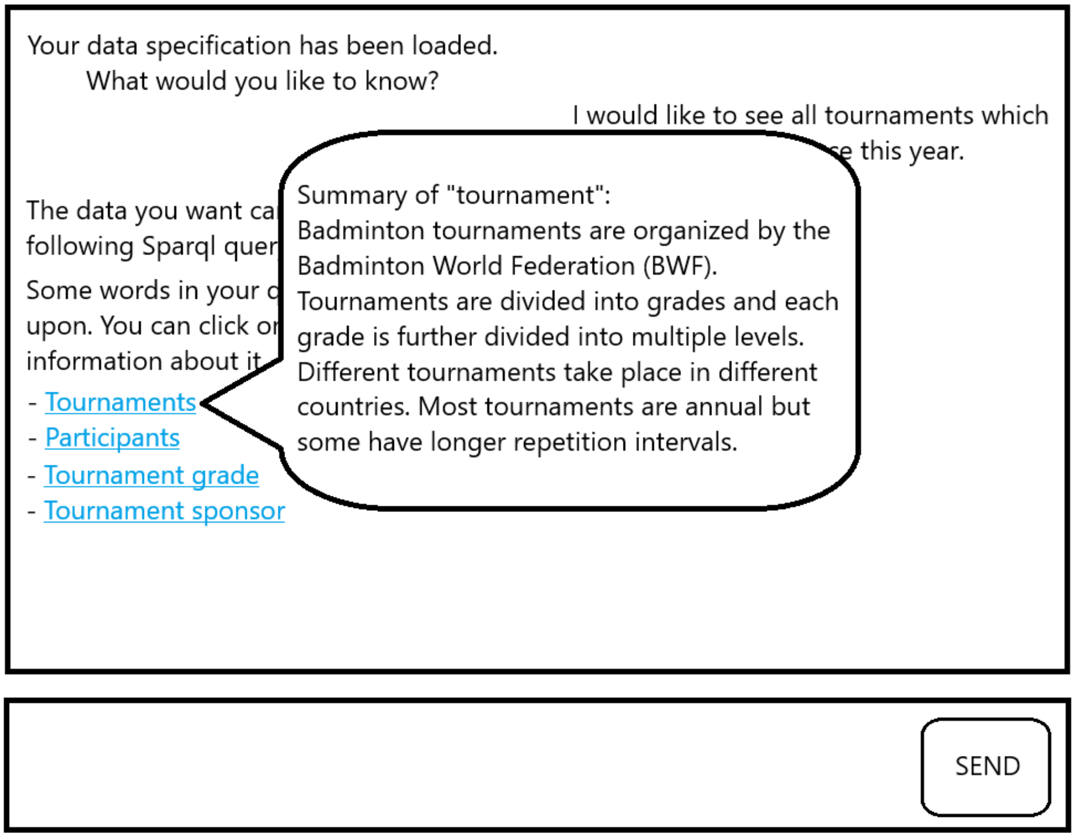
1. The user clicks on an item that is not yet in their question.
2. The user clicks on the button “Add item to my question” located in the item summary window.
3. The application shows a suggested message with the item added in the chat box.

Postcondition: The user has selected some items to add to their question and sees a suggestion in the chat box (*see Figure 8: Suggested message*). The user can now send the question as is in the preview or modify the question before sending it.

The user can choose more than one item to add to their question. Each time the user chooses an item the application will show a suggested message in the chat box with that item included. If the user chooses more than one item, the application will show a suggestion that includes all the chosen items.

Note that at this point, the user has selected some items to be added but they have not confirmed it, therefore the application is not doing anything except showing the message suggestion. The user confirms their choice of items by sending the suggested message into the conversation, after which the application will process the user’s message and then respond to it.

Figure 6: Summary of item tournaments



Your data specification has been loaded.

What would you like to know?

I would like to see all tournaments which

e this year.

The data you want ca

following Sparql quer

Some words in your q

upon. You can click o

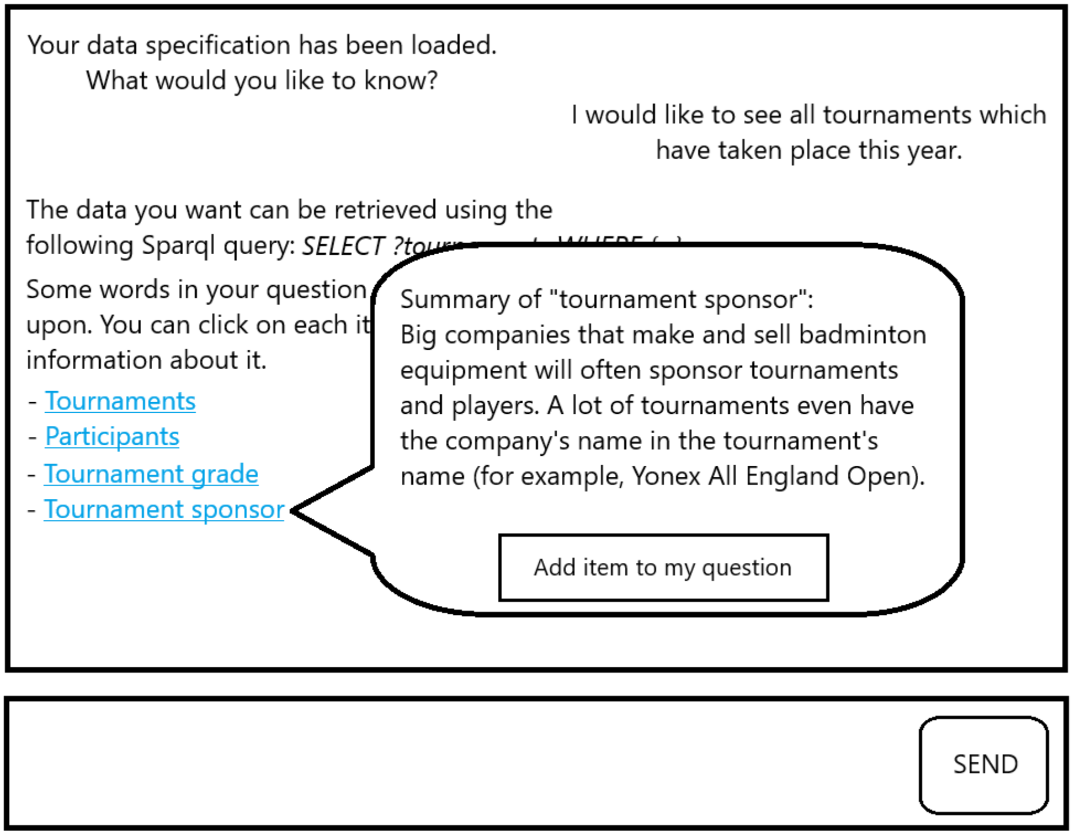
information about it

- Tournaments
- Participants
- Tournament grade
- Tournament sponsor

Summary of "tournament":

Badminton tournaments are organized by the Badminton World Federation (BWF). Tournaments are divided into grades and each grade is further divided into multiple levels. Different tournaments take place in different countries. Most tournaments are annual but some have longer repetition intervals.

Figure 7: Summary of item tournament sponsor



Your data specification has been loaded.
What would you like to know?

I would like to see all tournaments which have taken place this year.

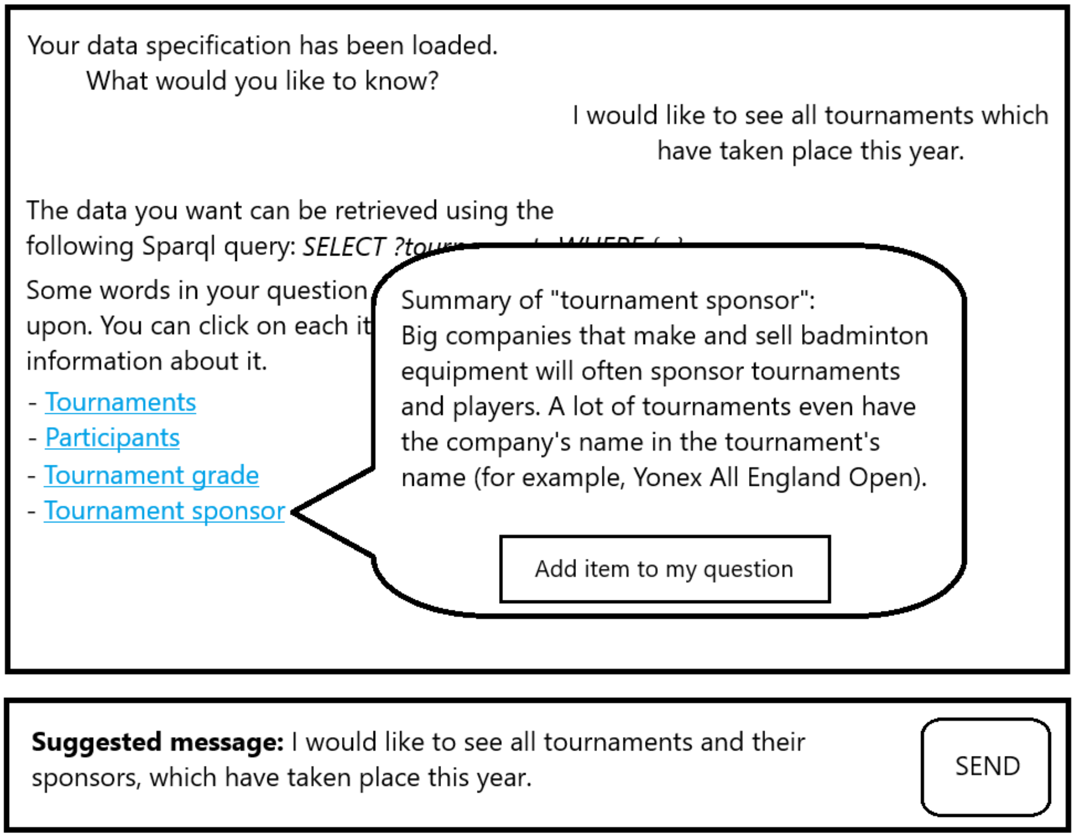
The data you want can be retrieved using the following Sparql query: *SELECT ?tour*

Some words in your question
upon. You can click on each it
information about it.

- Tournaments
- Participants
- Tournament grade
- Tournament sponsor

Summary of "tournament sponsor":
Big companies that make and sell badminton equipment will often sponsor tournaments and players. A lot of tournaments even have the company's name in the tournament's name (for example, Yonex All England Open).

Figure 8: Suggested message



Your data specification has been loaded.
What would you like to know?

I would like to see all tournaments which have taken place this year.

The data you want can be retrieved using the following Sparql query: *SELECT ?tou*

Some words in your question upon. You can click on each it information about it.

- Tournaments
- Participants
- Tournament grade
- Tournament sponsor

Summary of "tournament sponsor":
Big companies that make and sell badminton equipment will often sponsor tournaments and players. A lot of tournaments even have the company's name in the tournament's name (for example, Yonex All England Open).

Suggested message: I would like to see all tournaments and their sponsors, which have taken place this year.

5. The conversation model

This chapter discusses the model of the conversation between the user and the application.

5.1. Data specification

The user must provide a data specification that describes the data that they want to construct a query for. As mentioned in the Introduction, a data specification in this project will correspond to a Dataspecer package. The application will take an URI to the Dataspecer package and process it into a representation that it needs.

5.2. Data specification substructure

This is a subset of the data specification provided by the user. We get this subset when the user sends a question, and we map the question to some items in the data specification with the help of the LLM. We are building this substructure via a conversation with the user therefore one conversation is associated with one substructure.

As a side note, we could associate each user message to one substructure because the messages coming from the user are always fully formed question. Even if the user message is a question that expands upon a previous question, we can still consider it a standalone question, meaning if we create a new conversation with the same data specification and send the expanded question into that conversation as a new question, we should theoretically get the same substructure of the data specification as in the original conversation.

Associating substructures with conversations instead of messages is a design decision. As I have stated, because we are building the substructure via conversation, it makes sense to associate it with the conversation itself. When the user chooses some items to add to their question, we could simply add those items to the substructure, reinforcing the mental image that the conversation as a whole builds the substructure, not the individual messages.

5.3. Conversation

The user wants to get a Sparql query from the conversation with the application. The application builds a data specification substructure, and to do that it gets its inputs from the conversation with the user.

A conversation must always be linked to exactly one data specification that the user has provided.

A conversation contains all messages that have been sent into the conversation. This includes both messages from the user and messages from the application itself. I will describe the messages in more detail later in this chapter.

A conversation contains the data specification substructure that the application is building from the user’s choices. The first user’s question will give us an initial substructure if we manage to map it to some data specification items. Each subsequent message from the user will expand this substructure by adding more items to it, assuming that the user’s message expands upon their previous question. If the subsequent user’s message is completely different from their previous question, then the substructure will have to change, and we will start building it from zero. Referring to our example with badminton again, if the user first asks for tournaments and then chooses the item “participants” to expand their question, we can add that item to the substructure that we are building. But if the user first asks for tournaments and then suddenly writes a new question asking for “companies that sponsor players from the Czech Republic”, then it is completely unrelated to the first question, and we must discard the substructure that we currently have and start with a different one.

A conversation contains the suggested message to display to the user. If the user has not yet chosen any items to add to their current question, this suggested message will be empty (or null). To generate message suggestions, we will need to keep a copy of the current data specification substructure and whenever the user chooses one or more items to expand the question, we add the chosen items to this substructure copy. Then we use this copy in conjunction with the LLM to generate a suggested message for the user.

To summarize, each conversation contains the following:

1. A data specification (or a reference to the data specification).
2. Messages that have been sent into the conversation (from the user and from the application).
3. A data specification substructure that is being built from the user's choices.
4. A suggested message if the user has chosen one or more items to expand their question.
5. A copy of the data specification substructure with the extra items that the user has chosen to use for their question. This copy is used to generate the suggested message for the user.

5.4. Message

There are two main types of messages:

1. Messages from the user.
2. Messages from the application.

Messages from the application are further divided into three subtypes:

1. Welcome message.
2. Reply message.
3. Suggested message.

5.4.1. User messages

User messages are created when the user writes a message into the chat box and sends it or when the user selects some items to expand their question and then sends the suggested message (either as is or somehow manually modified).

Each user message contains a reference to one reply message from the application. Reply messages are discussed in subsection *Reply message*.

5.4.2. Welcome message

This is the first message in the conversation. After the application has loaded the data specification provided and created a new conversation, it sends a message prompting the user to write their question.

5.4.3. Reply message

A reply message is always associated with one user message and one user message contains a reference to one reply message. Reply messages never get created on their own. They are generated in response to an incoming user message.

If we fail to process the incoming user message, then the application will reply with a negative message saying that it did not find any suitable matches from the data specification and cannot help the user construct a query. Message processing can fail because the LLM cannot map the user's question to any items in the data specification.

If message processing is successful, then the reply message will contain the string to be displayed in the answer and a list of items related to the user's question.

5.4.4. Suggested message

This is a textual message without any additional data specification items, unlike the reply message. It is important to note that this message is not actually contained in the list of messages between the user and the application because it has not yet been sent into the conversation. The application generates this suggestion based on the user's choice of items to be added to their question, but only for the purpose of showing it to the user. If the user confirms their chosen items by sending the suggested message, then the application will add the message as user message.

6. Application architecture

This chapter discusses the architecture of the application which has a client-server architecture.

6.1. Front end

The front end is a web application running in the browser. It is a thin client whose main responsibilities are:

1. Display created conversations.
2. Allow conversation selection – to continue in a previously created conversation.

3. Allow deletion of conversations.
4. Let users send messages into the conversation.

6.2. Back end

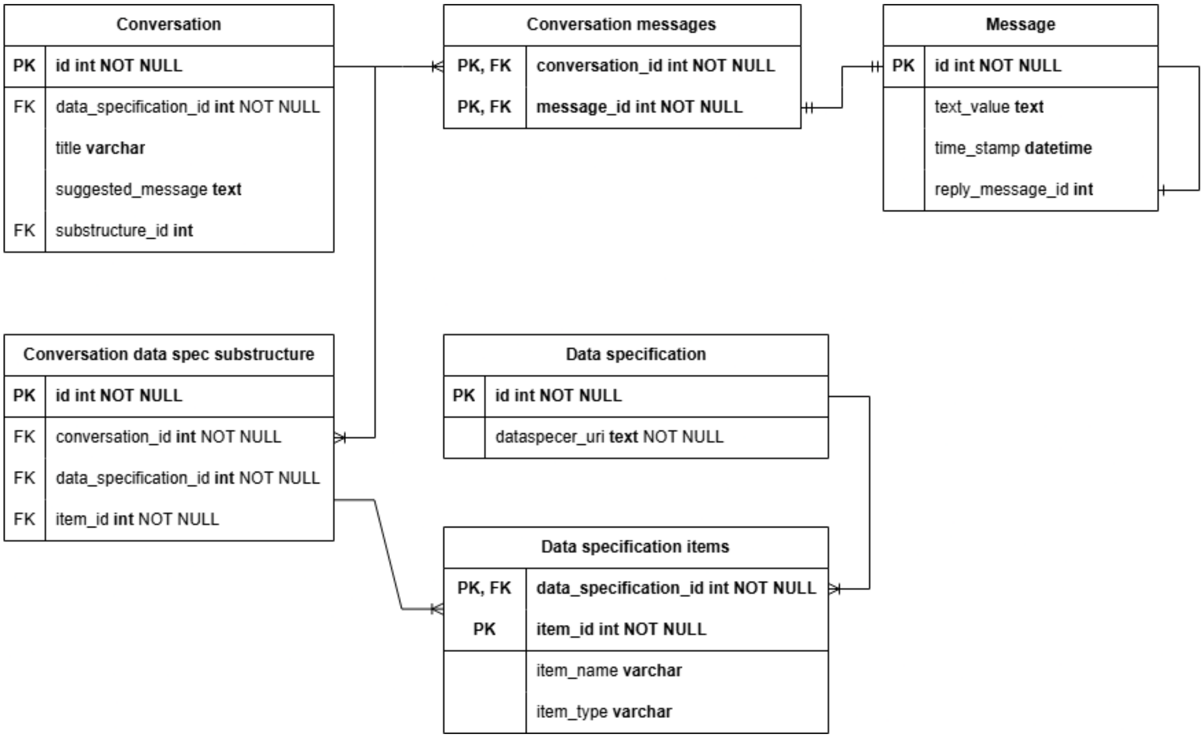
The back end can be separated into layers and each layer contains some modules. These layers and modules help achieve separation of concerns. Figure 10: The application architecture shows the layers and modules of the architecture. Each rectangle represents a module with its own responsibilities while the rounded rectangles inside some modules shows notable parts of that module.

6.2.1. External systems layer

The LLM and Dataspecer are systems running independently of this application. I have included the database into this layer not because it is an independent system per se, but services do not send direct SQL queries to the database. Instead, they will utilize some framework, which can be viewed as a connector to the database.

The database is a SQL database where the application will store data specifications and their items, conversations and their messages and the data specification substructures built from the conversations.

Figure 9: Database schema



6.2.2. Connectors layer

This layer contains connectors to external systems.

The LLM connector is the most notable one. This connector helps ensure independence of any specific LLMs or LLM framework. Different LLMs might require different prompting strategies and different ways to communicate with them so switching to a different LLM would mean supplementing the implementation of the LLM connector for that specific LLM. This module has two notable parts which are the prompt constructor and the response processor.

The prompt constructor combines the user's question and its own prompt templates to either get the items mapping from the question or get the items to recommend to the user.

The response processor parses the LLM responses into data classes for the services in the core layer.

The dataspecer connector retrieves Dataspecer packages from a given URI.

The database connector constructs and sends queries to the database.

6.2.3. Business core layer

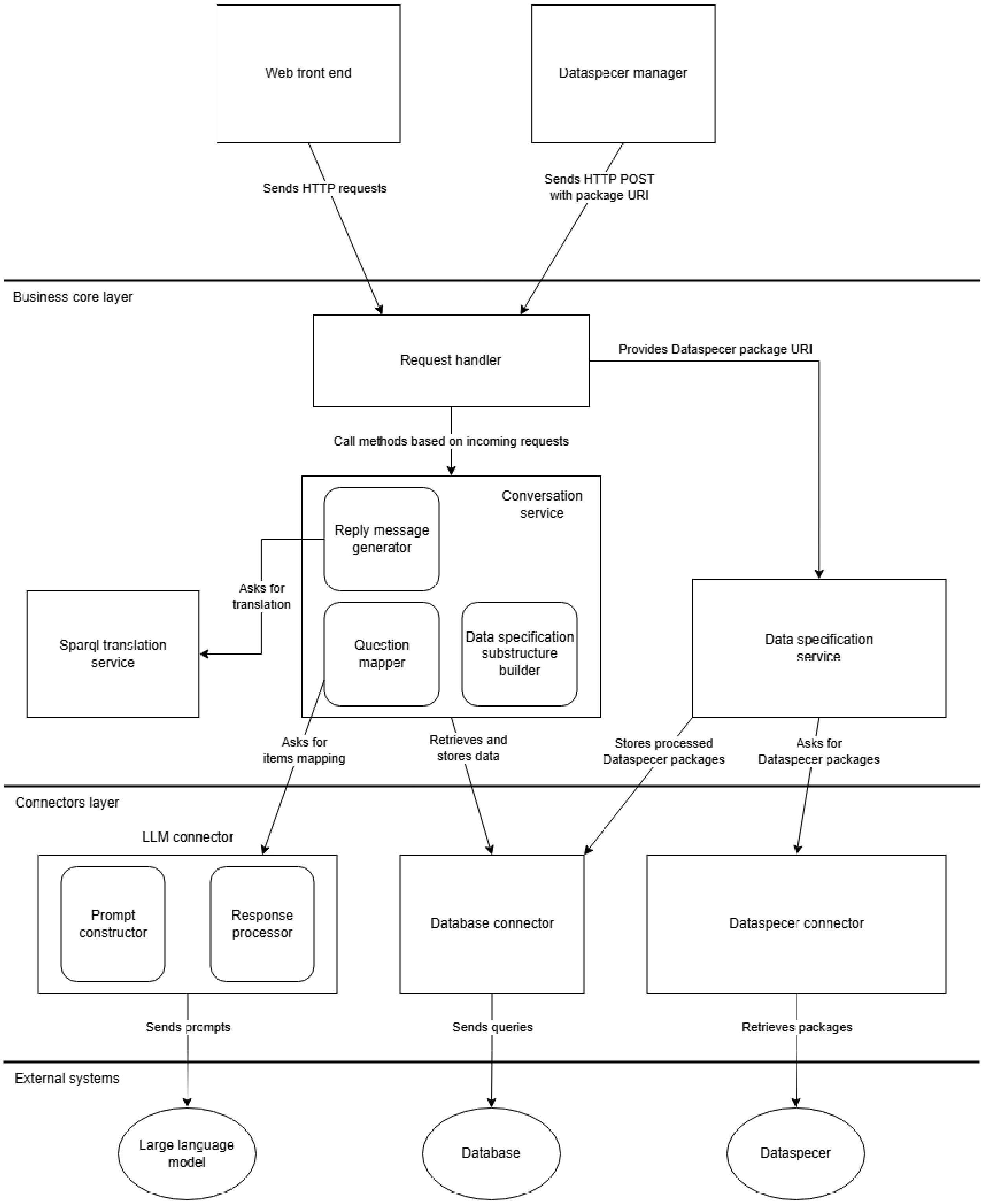
This is the layer that the front end communicates with. Incoming requests are intercepted by the request handler, which then delegates whatever work that needs to be done to other modules in this layer by calling the appropriate methods.

The data specification service calls the dataspecer connector to retrieve the Dataspecer packages that it needs and processes the package into an internal representation that the application needs.

The conversation service handles all operations that involve conversations. It creates a new conversation, adds new messages to the conversation, generates reply messages and suggested messages for the user. It also builds the data specification substructure from the conversation.

The Sparql translation service generates Sparql queries from the substructure that the conversation service is building from in the conversation.

Figure 10: The application architecture



7. Technologies

As recommended by Štěpán Stenchlák, the front end will be built using Vite, React and Typescript and Shadcn/ui as a graphical library.

The back end will be written in C#, utilizing its Minimal API feature to build a Restful back end.

8. Project time schedule

I have didived the project's development into 5 phases.

8.1. Phase 1: October 2024 – November 2024

During this period, I tried prompting a few different LLMs to explore their capabilities and see if they give reasonable responses that can be used for the project. The results seemed promising. I submitted the research proposal on the 27th of November 2024.

Estimate: 100 man hours.

8.2. Phase 2: December 2024 - April 2025

During this period, I came up with suitable user stories and use cases for the project. I analyzed the use cases and proposed a conversation model,architecture and the back end API. I also implemented a mock back end to showcase the API and use cases to the team.

Estimate: 200 man hours.

8.3. Phase 3: May 2025

The main goal in this phase is to figure out, how to represent data specifications and data specification items in my application. To that end, I will experiment with the LLMs deployed on the KSI server to see what I need to give them from the data specification in order to get the output that I desire. I will also figure out suitable prompt templates and save them for the first back end implementation. I will fine-tune these prompts in a later phase of the project.

Estimate: 100 man hours.

8.4. Phase 4: June 2025 - July 2025

I will implement a funtional back end that is connected to all external systems. I will then implement the front end that communicates with the back end.

Estimate: 300 man hours.

8.5. Phase 5: August 2025

I will fine-tune the prompt templates to get optimal results. I will then test the application against a select few LLMs and compare the results.

Estimate: 50 man hours.