FOR SPRINT 1:

**Edit Food Post so that it has new attributes:**

* Timestamp at which it was posted
* Start Date for pickup
* End Date for pickup
* Start Time for pickup
* End Time for pickup
* Username of who posted it.

PS: Start Time and End Time apply to every day between and including the start date and the end date.

This iteration was started conceptually, on the database side. So I started by expanding on the database diagram I had in Astah. The relevant part is as follows:

Diagram

Description automatically generated

Text

Description automatically generated

Here, I also present the SQL code I added to the data definition file. I altered the table instead of dropping it and recreating it since this would involve the loss of previously existing data.

A good way to start on Java could be to edit the FoodPost class, so I added the new attributes there and mapped them to the new columns accordingly:

Text

Description automatically generated with low confidence

I then also adjusted the constructors and getters.

The FoodPostCreationDTO also had to be edited (the new attributes added).

Table

Description automatically generated

The FoodPostService (implementation) class had to be changed slightly, in that I had to parse the new attributes from the DTO to the constructor of the Food Post created.

Graphical user interface, text, application

Description automatically generated

What’s to note here is that a *post state* is automatically “posted” as soon as it’s posted, the timestamp for creation is defined in this layer (as the present moment) and the user is gotten from the database before it is put into the Food Post, and if it is not found, an exception is thrown right here. Then, if everything is okay, the Food Post is saved by the Repository class in the database.

The next step was to think about where this method would be called – and the answer was gRPC. So it was time to change the proto file for the Food Post.

Graphical user interface, text

Description automatically generatedHere is what the Food Post Request looks like. It “mirrors” the Food Post CREATION Dto, in that it has exactly the same fields.

The types “Date” and “Time” were manually defined in the proto file as well, both with the simple and expected fields (day, month, year; hour and minute).

Graphical user interface, text

Description automatically generated

And here is the Food Post Response, which corresponds to a “whole” Food Post object. There was a HUGE struggle regarding the Timestamp field because the import from Google Protobuf was not working. So I had to make changes to the .pom file (with trial and error…) until it eventually worked out without breaking everything. I could have also just defined a Timestamp myself lol, but that didn’t cross my mind.

The services stayed the stame. So I “cleaned and installed” the new grpc changes and moved onto updating the method in the grpc service class. This was a big boy.

Graphical user interface, text, application

Description automatically generated

Looking back, I probably could have moved all this conversion to the Converter class created by Kamil; but that also did not cross my mind. It is something to improve later, maybe.

That is it for the Java part. Here is also a screenshot of BloomRPC showing that a request made from GRPC makes its way to the database and returns the full object.

Text

Description automatically generated

Now, for the C#.

When I do Java, I start from the database (which is one end of the application), and when I do C#, I tend to start from the page/blazor/whatever, which is the other end. And then I, lastly, “join them in the middle”. So I started from the “CreateFoodPost.cs” page file, where I added input fields for dates and times according to the new fields.

Graphical user interface, text, application, email

Description automatically generated

Before moving onto the code of the page, I had to change the Domain classes – the Food Post and the Food Post Creation DTO.

Below is the Food Post. I just added the new fields. I also adapted the constructor.

Graphical user interface, text

Description automatically generated

First, I had used the classes “DateOnly” and “TimeOnly” from C# for the Start and End Dates and Times because they represented exactly what I needed, but they caused me A LOT of trouble because the C# Json Serializer could not serialize them for some reason. I don’t know how, but I was stuck trying to find a fix for maybe three hours… And every time, it either didn’t work or I broke the whole system. All of this when I could have just opted for an alternative, such as making my own classes for Date and Time… (which is what I eventually ended up doing, thanks to the suggestion from Kamil. And it didn’t even take that long. Grr!) Anyways.

Having changed the Domain classes, I could go back and change the code for the page.

Table

Description automatically generated with low confidence

I added the new fields.

Graphical user interface, text

Description automatically generated

When opening this page, I set the creator Username to Kamil by default, because this is a user that I know exists. The reason for hardcoding a user is because it is not my task to attach the currently logged in user to the creation of a food post.

The method “initDates()” just sets both dates for today and both times for 00:01.

Then, the CreateAsync() method, which is the star of the show.

It starts by converting the DateOnly and TimeOnly objects that are bound to the input fields to the MyDate and MyTime classes created by me, so that these variables can be put inside the DTO to be sent to the gRPC client. I then create a DTO and call the method “Create” on my Food Post Http Client. That is it for the Blazor page.

Graphical user interface, text

Description automatically generated

The Food Post Http Client suffered no changes, because all it does is get the DTO and post it to the Web API endpoint to create a Food Post. No changes were made in the Web API. The Web API calls the “create” method from the Application/Logic layer, which also suffered no changes. The reason why none of these layers suffered changes is because they call methods with the Food Post Creation DTO, which was altered in an extendable way so that these “intermediary” layers would not be affected. The Application layer then calls the Food Post DAO, which deals with gRPC and therefore needed changes.

So I changed the proto file to look exactly like what it did in Java, rebuilt the solution and dived into the gRPC Food Post DAO.

The Create method changed exactly in the way you would expect: what it does is create a Food Post Request gRPC object, post it to gRPC, return the response and make a Food Post Response object out of it.

However, this method takes a Food Post Creation DTO, so it first needs to convert it to a Food Post Request gRPC object. This is lengthy, but luckily, not tricky. It is very straightforward and the next screenshot will illustrate that.

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

It’s that simple.

And that is it for altering the Food Post object with new fields. I do not want to text because I’m sleepy, but I am aware that only the sunny scenario will work and that there is barely any error handling – this should happen in the Application layer, which should present the user from making silly things like making the End Date be before the Start Date or the End Time before the Start Time. Other than that, I’d say it’s pretty solid. This was so frustrating lol, but it taught me to think of alternative solutions before going into a wormhole (?) rabbithole? of the same possibly-helpless solution.