Is it possible to make a Flow tool by using ThingsDB?

Introduction

In this document I will be researching if it's possible to make a flow tool by using ThingsDB. This research question is important for this internship because if this research comes out negative it means that I can't continue with what I'm working on. This document will first of all include research on what the flow tool should include, if ThingsDB gives solutions to realize these and in what ways ThingsDB needs changes or additions to make a flow tool possible. That means that this document has several researches that have been done to form this document. At the end of this research I will know if I will be able to make a flow tool using ThingsDB and I will have a lot more insight on how the flow tool should look like.

Methods I used for this research

For this research I used several methods to find out if it would be possible to have a flow tool on top of ThingsDB, as this is one of the most important research questions to find out, as this controls most of my internship, I did a lot of research. First of all I did some research on my own using the library method as I needed to find out what ThingsDB was and how it worked. Then I asked my stakeholders by interviewing them what kind of must haves the flow tool should have. Then I researched how these must haves could be interpreted in the flow tool that I will be building by doing research on ThingsDB itself and after that we sat together again to make a probability score to make a prediction on how easy things would be to make and if it would be achievable in my internship period. Finally I made a prototype myself that connects a front-end with the ThingsDB back-end to confirm that it is indeed possible to connect the two.

Research on the requirements

First of all I have to do research on the parts that the flow tool should at least include. To do this I will be using the literature study, as I will be taking a lot of inspiration from other flow tools (which are linked in the sources area). I also did a field study asking people in my company what the tool must have, as I'm making it for them it is also important to hear about their opinions and knowledge about the tool. Below I have listed the must haves of the flow tool based on both studies done.

Requirements the flow tool must have

- A component needs to handle a custom code that's assigned to it
- Errors should at least go to the either the log, mail or DutyCalls
- A component can be filled in with extra information about the procedure
- All saved flows can be seen in a list and should show if it's running or not, or isn't done yet
- The flows should have a begin, middle and end component
- Data can be changed, set or removed by means of a flow
- Components need to transfer the data they contain to other components
- You need to have the option to save flows and use them as a component
- The end component should be an error/success message
- An error/success component can be templated and should be reusable
- There should be a playground where you can test specific components or flows (Optional)
- Components should have an easy to follow documentation for users
- Components should be intuitive
- Flows must be high available
- A flow needs to be able to schedule
- Long running component (micro service)

Does ThingsDB give options for these must haves

To see if ThingsDB can be used for the flow tool I will go through all of the must haves that we have just created and look if ThingsDB has the potential to make the flow tool run. By doing this we can make sure that ThingsDB will be viable enough so we are able to make a flow tool on top of this.

A component needs to handle a custom code that's assigned to it

ThingsDB has a function called procedures, you can make a new procedure with a variable that still needs to be filled in. see the example below;

```
// Create a new procedure `add_one`
new_procedure('add_one', |x| {
    "Adds one to a given value";
    x + 1;
});
```

In this example X still needs to be filled in, this means that in the front-end you can customize a component and that this can be handled in a procedure in the backend.

Errors should at least go to the either the log, mail or DutyCalls

When an error occurs on the front-end it is important that the user knows this, so the error should be sent somewhere so that the user is informed and will know an error occurred on the flow. ThingsDB made a module for DutyCalls that is integrated already. This means that an error can be sent to DutyCalls already. Logging can also be done in either the front- or back-end. And lastly the mail isn't yet compatible with ThingsDB so a module for this still needs to be made, so this is still something that should be worked on so the user can receive the error via their mails.

A component can be filled in with extra information about the procedure

Same for the first must have this can be executed with the procedures, you can just simply fill in a variable which can then be used for the rest of the code. Here is an example that gives a pretty good view on how procedures work;

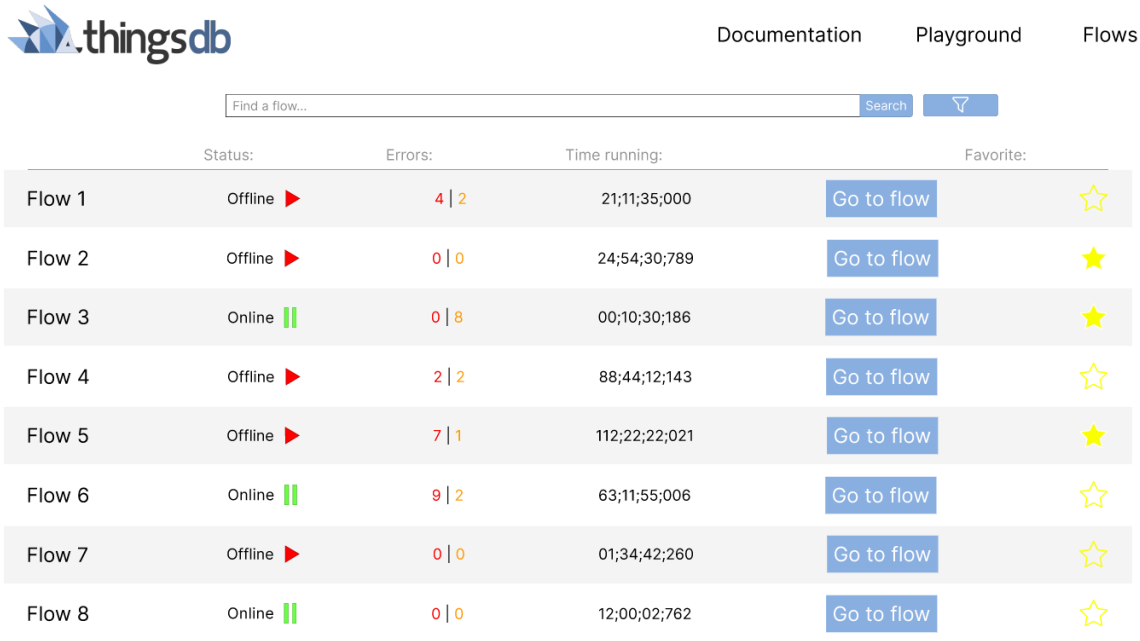
```
// create a greet procedure
new_procedure('greet', |name| is_str(name)
    ? "Hello " + name
    : "Hello unnamed user!"
);

// run `greet` with a given name
greet_iris = run('greet', "Iris");  // equal to: greet("Iris");

// run `greet` with nil
greet_nil = run('greet', nil);  // equal to: greet();

// return the greet response values
[greet_iris, greet_nil];
```

Return value in JSON format

```
[
    "Hello Iris",
    "Hello unnamed user!"
]
```

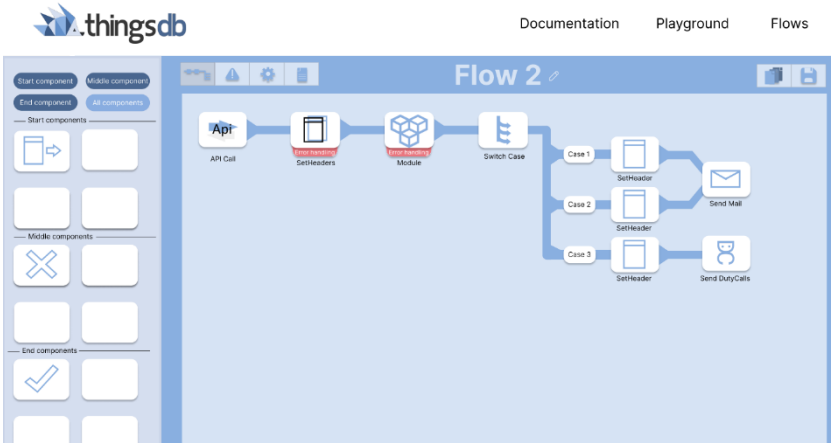
All saved flows can be seen in a list and should show if it's running or not, or isn't done yet

The flows can be saved in the database with the additional information that will be needed, such as if he is running, if the flow is favored etc. The flows can then be retrieved with a GET API call and be showcased on a webpage. (The picture below is a mock-up)



The flows should have a begin, middle and end component

The components have different functions and need to be separated, because some components should only be used as the beginning component and some should only be used as the last component. The last component will have the same functionalities as the error components that I have mentioned earlier. This component needs to let the user know if the flow succeeded or failed and this can be done via log, mail or DutyCalls. The first component will be an API call or a JSON file, these can be handled in ThingsDB as well and the middle components will include things like switches and converters.



Data can be changed, set or removed by means of a flow

When a query uses a statement which makes a change to ThingsDB, then internally ThingsDB will create a change with a new change-Id to apply these transformations. Changes are applied in order on each node; so database consistency is guaranteed. A single query might contain several statements and make many changes. All changes within a query will be grouped in a single change with the exception of future callbacks. Each future callback will get its own change-Id if a change is required.

Components need to transfer the data they contain to other components

When submitting code in ThingsDB you can choose what kind of answer you want to receive, TREE, JSON, Arguments and Logging. This means that there are many ways to receive and send through the data that has been made by components. To transfer this data to other components you simply have to retrieve the information and put it in the lacking information in the component that's needed. This could also be done with Cookies, so you can save the information in your browser and put it in the components that way. This could cause some errors though. (Source 8.)
A few disadvantages of cookies are;

- Browser Impacts. Cookies are not restricted based on internet usage.
- Security Risks. Since cookies are stored in the hard drive as text files, it posses some serious security risks.

- Size Limitations. Size limitations also exist on cookies.
- Privacy Concerns.
- Manual Disabling.
- Encoding Information.

You need to have the option to save flows and use them as a component

Again this could be done by saving the flow as a procedure, the code will then be saved as a procedure and can then be used as a component. Of course we still have to think about adding a component for it and things as adding a picture to that designated component. But that is more front-end based and can definitely be done.

The end component should be an error/success message

As stated earlier there is a module for DutyCalls already, so the component will be able to be made with Logs and DutyCalls. This does mean that ThingsDB still needs the module for mailing. We still need to think of a way so that the end component can only be put to any of these three, but this can surely be done by using restraints in the front-end.

An error/success component can be templated and should be reusable

It is possible to make a template which can be filled in by means of a procedure, these can be stored in the backend so the information can be reused for other flows/components. How will this work? In the front-end you'll be able to configure the account you want it to go to for an e-mail or DutyCalls, for DutyCalls you'll also be able to choose a channel that you want to send it to. These components can be reused so that you don't have to fill in the same error/success component for every other component.

Components should have an easy to follow documentation for users

What we came across is that the documentation for most of the flow tools are very developer based. Many tools shout that they are very user-friendly and made for non-developers, but when it comes down to the documentation of specific components it is very developer-based. We find it important to make documentation that is very clear and written in 'Jip-&-Janneke' language so that people will actually understand what the component is for and how they are supposed to use it.

Components should be intuitive

Components should already know what they have to do for the most part, some parts have to be filled in still by the users. Given that ThingsDB has procedures and tasks will help with components being intuitive, as what has to be done will be saved and can be used by filling in the missing information that's still needed.

Flows must be high available

ThingsDB works with nodes, you can add as many nodes as you like. When something happens one node goes down while the other ones stay running, when the first node is done the other ones will follow and do the procedure as well. This causes for ThingsDB to be highly available and have almost no down time. This is great for the high availability and this will work hand in hand for the availability of the flow tool.

A flow needs to be able to schedule

ThingsDB works with tasks, you can schedule specific tasks that you want to happen on specific dates/times. This is convenient for the flow tool as we can make tasks and schedule flows. You can do several things with the task, which you can read in the docs page from ThingsDB.

Long running component

There could be a possibility that there will be a long running task in a component that, for example, takes an hour to complete. This could cause problems because ThingsDB can have multiple nodes, which can go down one by one to make backups or to restart and the other nodes can still run while this happens so you don't have downtime. If you have a long running task, such as a download, this means that the node can't go 'offline' for a long period of time so this could give a lot of problems with the node and destroy the reliability that ThingsDB wants to give the users.
This means that we still need to find a solution for these long running components, because ThingsDB wants to keep the reliability and high availability. We ran into this problem fairly

early and for this I asked my stakeholders for a brainstorm session for how we could fix this so that it would work with ThingsDB. For this I used the Workshop and Library method, as we did a literature study, expert interview, sketching and ideation for this brainstorm session. We had a very long whiteboard session with Rik and Jeroen to come up with some ideas that could solve this problem that we had.
First of all we came up with the idea to make the long running components pausable, but we scratched this idea pretty early on as this is really hard to make and we also had to think about an idea that was achievable in the certain space of time we had. Then we came up with the idea to not do anything about the long running component and just let it be, but this wasn't really the way we liked it either. Then we started thinking outside of ThingsDB, we asked ourselves what if we build something outside ThingsDB that will check if the long running component is still running, and if it doesn't it will give back an error. So we decided that we could create a micro service in between ThingsDB and the flow tool and that we could create a 'await until' component that has a 'heartbeat' to check if the component is still running with an API call. If we make this this could solve the problem with the long running components and this means that there would be no problem having these in the flow tool.

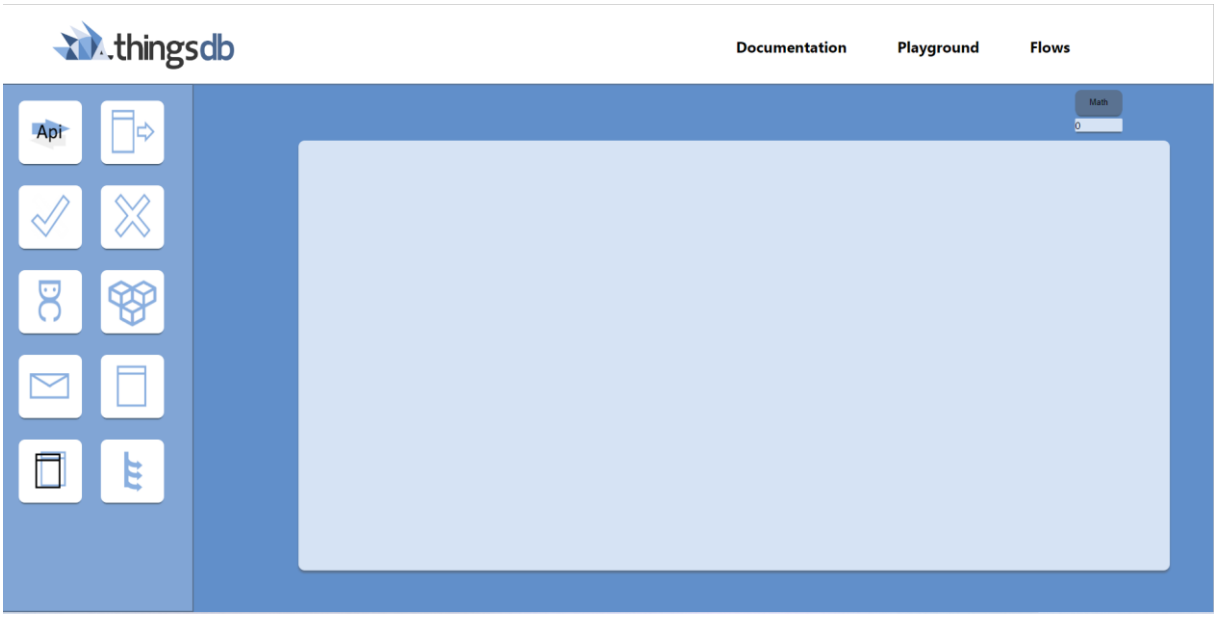
Probability score

Must haves	Probability score (0/10)
A component needs to handle a custom code that's assigned to it	0
Errors should at least go to the either the log, mail or DutyCalls	0
A component can be filled in with extra information about the procedure	0
All saved flows can be seen in a list and should show if it's running or not, or isn't done yet	0
The flows should have a begin, middle and end component	0
Data can be changed, set or removed by means of a flow	0
Components need to transfer the data they contain to other components	0
You need to have the option to save flows and use them as a component	0
The end component should be an error/success message	0
An error/success component can be templated and should be reusable	0
There should be a playground where you can test specific components or flows	0
Components should have an easy to follow documentation for users	0

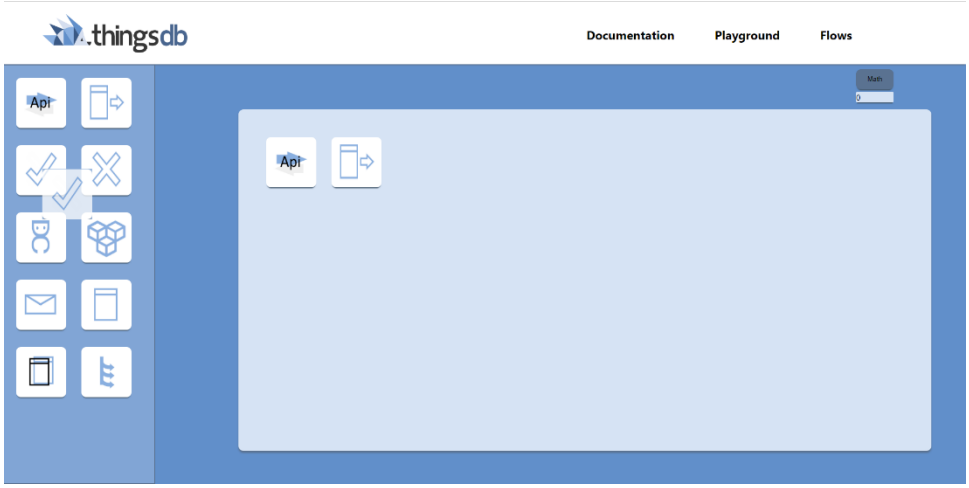
Components should be intuitive	0
Flows must be high available	0
A flow needs to be able to schedule	0
Long-running components should not interfere with the ThingsDB away mode functionality.	0

Prototype

To finish this research and know if it is really possible to make a flow tool on top of ThingsDB I made a little prototype that shows it is indeed possible. First of all I made the structure of how I wanted it to look based on the research and sketches I made on Figma.

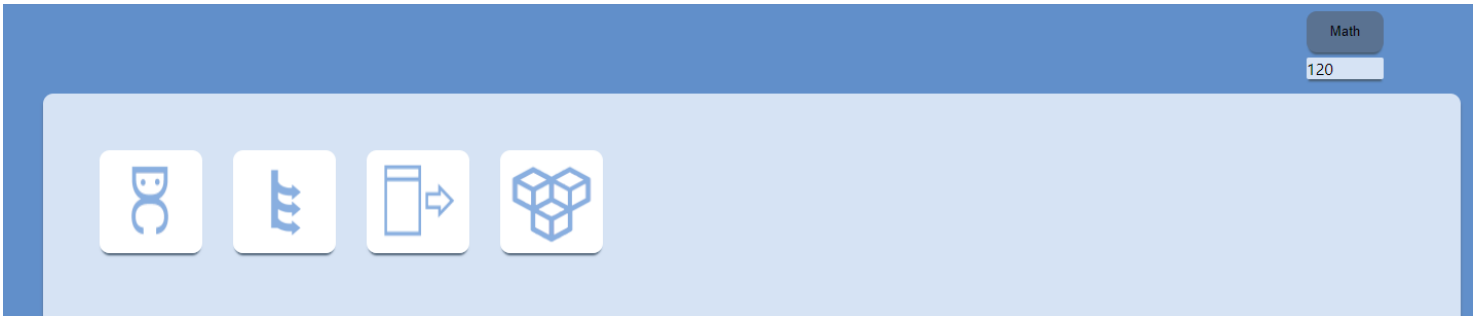


The fact that I made sketches on Figma helped me structuring everything more easily so this caused me to finish this earlier. Then I needed to make a connection with the backend, I wanted to do this by dragging the components in already but with a simple connection to the back-end that is easy to explain.

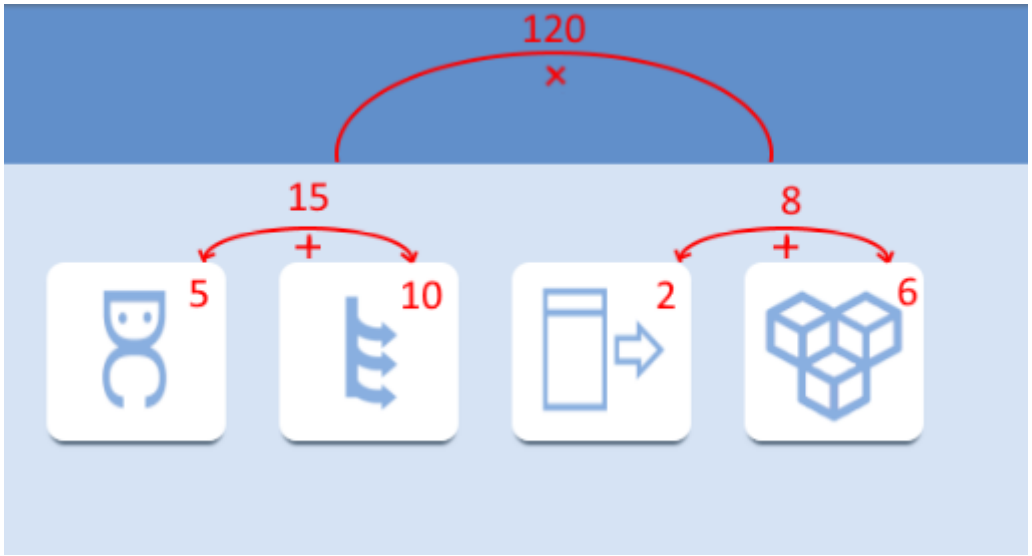


As you can see the items can be dropped on the board by dragging them inside, the dragging is visible on this picture on the left hand side where the checkmark component is doubled and more transparent (the transparent one is currently being dragged).
So this is where the connection with the button comes in. Every single component on the sidebar has a value, from up to down they have a value of 1 to 10 (as stated down below).

When you drag some components on the board and u press the math button on the right hand side you get a value, in the next example you get the value 120 with the four components that you have dragged in.



How did the API come up with the number 120? Well, I will show you with a little example. First of all it splits the board, and it grabs the values of the components and adds it together. Then it grabs both of the numbers that you got from adding the components and it multiplies them, as shown below it shows how the API gave back number 120.



Well how does this work underwater? First of all I made a list of components that all have a value with them.

```
const pictureList = [
    {
        id: 100,
        url: './Images/ApiCall.png',
        int: 1,
    },
    {
        id: 200,
        url: './Images/designsFigma.png',
        int: 2,
    },
    {
        id: 300,
        url: './Images/designsFigma2.png',
        int: 3,
    },
```

When pressing the button you go to the ApiCall function where you first go through a little equation that calculates the first and the second half of the board and also adds them together.

```
const half = Math.ceil( x: board.length / 2);

const firstHalf = calculateInt(board.slice(0, half));
const secondHalf = calculateInt(board.slice(half));
```

```
const calculateInt = (halfPictures) => {
    let halfInt = 0;
    halfPictures.map((picture) => {
        halfInt += picture.int
    })
    return halfInt
}
```

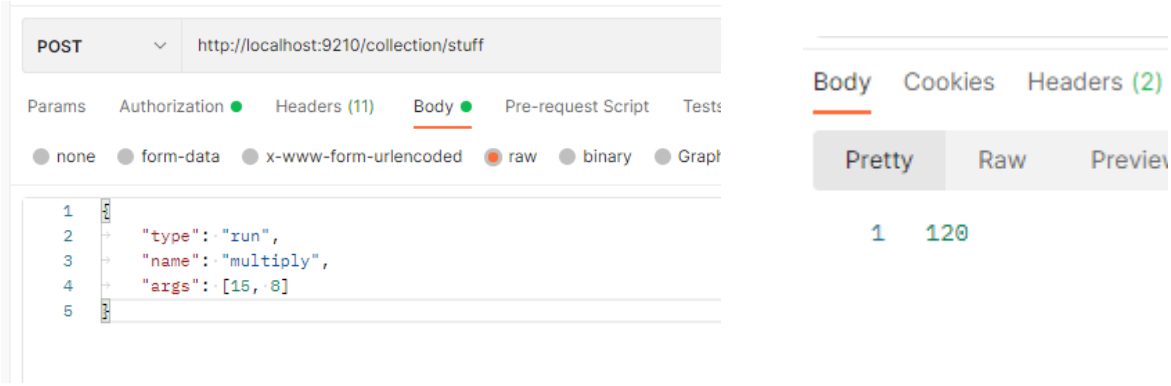
Then I made a axios post that sends the information to the back-end and with the argument I choose what kind of procedure has to be done with the given information.

```
    axios.post( url: "http://localhost:9210/collection/stuff", data: {
            'type': 'run',
            'name': 'multiply',
            'args': [parseInt(firstHalf, radix: 10), parseInt(secondHalf, radix: 10)]
        },
        config: {
            headers: {
                'Authorization': `Bearer ${token}`,
                'Content-Type': 'application/json'
            }
        })
        .then(res => {
            setAnswer(res.data);
        })
}
```

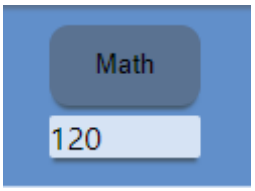
As you can see the procedure that has to be done is to multiply, this is a procedure that I have made already in ThingsDB that looks a little like this;

```
1
2   new_procedure('multiply', |a,b|){
3       a * b
4   }
```

First of all I tested this in Postman, I filled in the required information and as you can see the same answer returns.



When this is done the answer returns and gets set in the answer component. This is the part underneath the button where the number 120 was shown.



And that summarizes how I made a prototype by making a connection between the front- and back-end.

Conclusion

If we summarize all the information we got out of this research we can conclude that it is definitely possible to make a flow tool on top of ThingsDB, further research has to be done to know how these conclusions will work in practice but that's for another research document. This research has given me more knowledge on ThingsDB, what kind of must haves the flow tool should include and how I want the flow tool to look like in general. This research will help me on the journey as this gives more context on the project itself and this is kind of the frame I will be working from further on. My stakeholders think it's good to brainstorm about the context of the project and write it down so that you'll always know what the plan is, and that it's clear enough you'll still know what to do in five years.

Sources

1. https://www.servicenow.com/nl/now-platform.html
2. https://www.xmatters.com/features/flow-designer/
3. https://messagetothemoon.nl/telefonie/hosted-3cx/call-flow-designer
4. https://www.mulesoft.com/platform/api/flow-designer-integration-tool
5. https://powerautomate.microsoft.com/nl-be/
6. https://docs.thingsdb.net/v1/
7. https://thingsdb.net/
8. 6 Advantages and Disadvantages of Cookies | Limitations & Benefits of Cookies on Website (hitechwhizz.com)
9. Jeroen van der Heijden, Stakeholder
10. Rik Lempen, Stakeholder