Flow BP

Software Engineering Project
Application Design Document

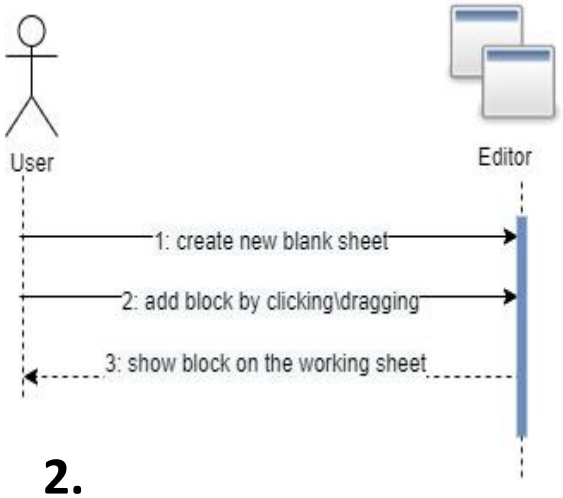
Tomer Bitran
Shir Markovits
Shahar Hazan

Contents

Chapter 1- Use Cases

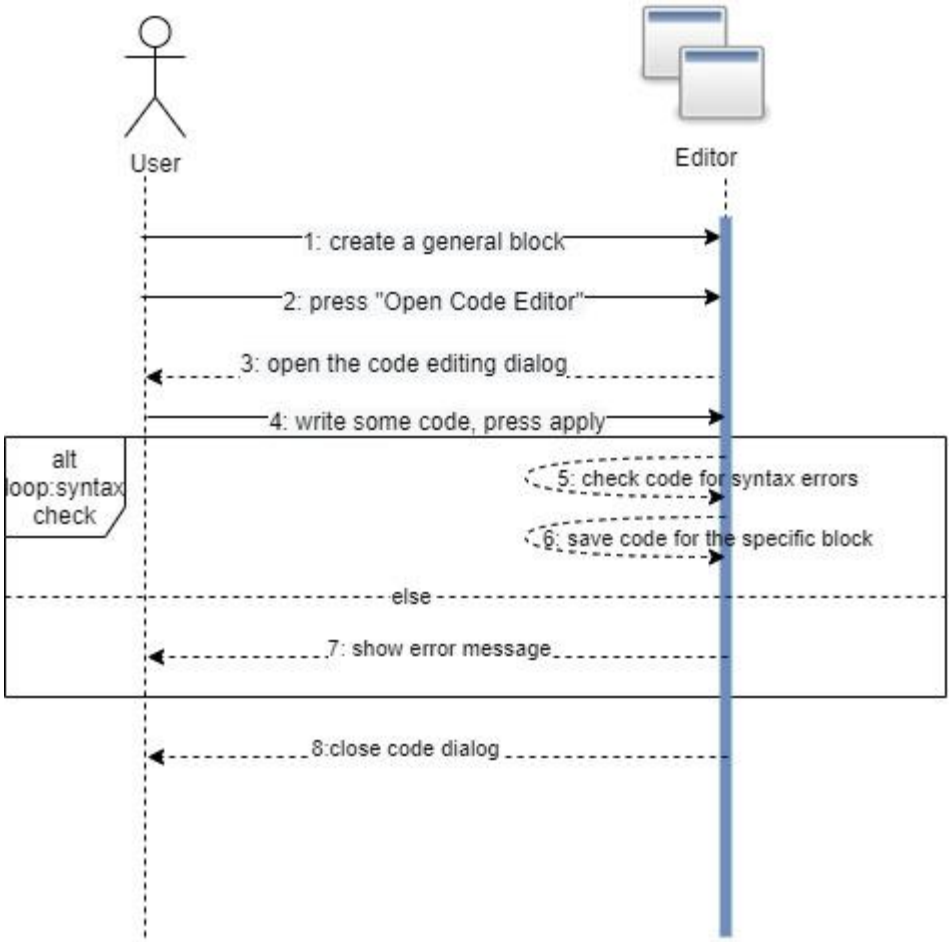
1.

Use-case name	Block addition
Description	The user opens the blocks pop-up menu by clicking the right mouse button and chooses a block.
Pre-conditions	None.
Post-conditions	The block will be visible on the working sheet.



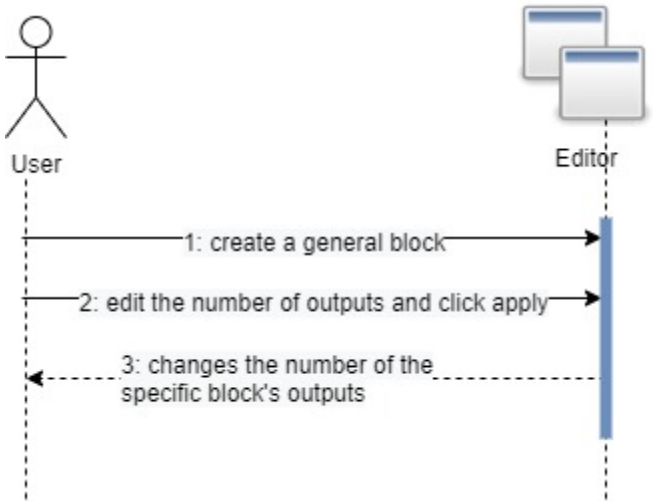
2.

Use-case name	Edit code section on a General/Console block
Description	After creating a general/console block, the user can edit the code on the block by clicking the block, then clicking the "Open Code Editor" button. After clicking the button, a new window with a text area will be opened, and in order to save the code the user should press the apply button.
Pre-conditions	A general/console block was created on the working sheet.
Post-conditions	The editor saves the code of the specific block.
Alternatives	If the user writes code with syntax errors, the editor will notify the user that an error has occurred, and allow the user to fix the code.



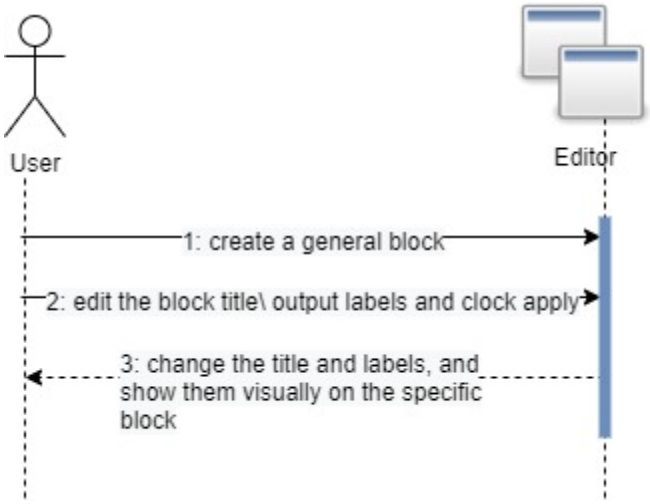
3.

Use-case name	Edit Number of outputs on a general block.
Description	After creating a general block, the user can edit the number of outputs the block has by clicking the block, then editing the "Number of outputs" field. Then pressing apply will change the number of outputs.
Pre-conditions	A General block was created on the working sheet
Post-conditions	The number of outputs on the specific block is changed accordingly



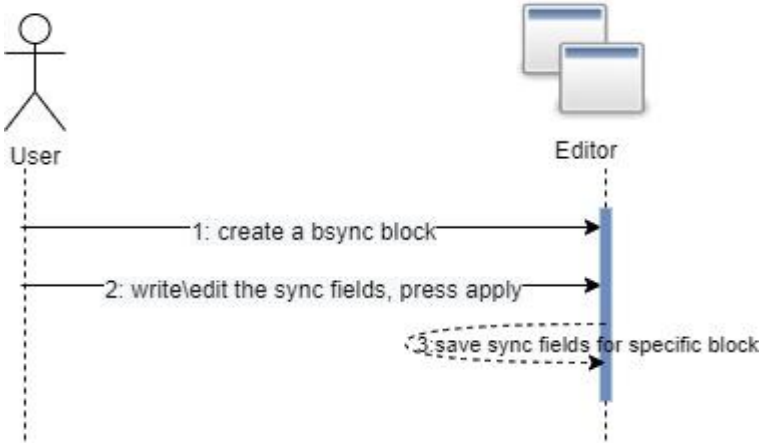
3.1

Use-case name	Edit outputs labels \ block title on general block
Description	After creating a general block, the user can edit the output's labels and the block title by pressing the block and changing the field.
Pre-conditions	A general block was created on the working sheet
Post-conditions	The labels\title are showed visually on the block



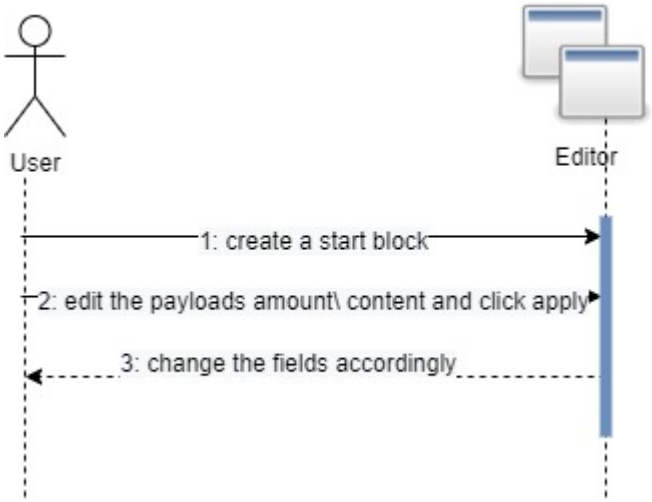
4.

Use-case name	Edit Request, Wait and Block fields on a Bsync block.
Description	After creating a Bsync block, the user can edit the Request, Wait and Block event fields on the block.
Pre-conditions	A Bsync block was created on the working sheet.
Post-conditions	The editor saves the sync fields of the specific block.



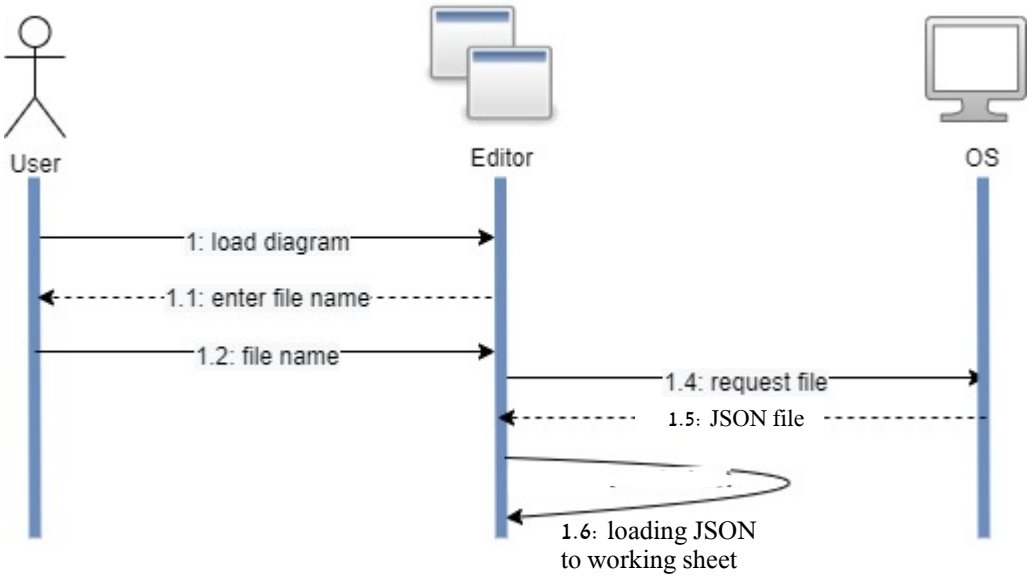
5.

Use-case name	Edit Payloads on a start block.
Description	After creating a start block, the user can edit the number of payloads the block contains, and the payloads contents. The user clicks the block and edits the fields on the pop-up window.
Pre-conditions	A start block was created on the working sheet.
Post-conditions	The fields on the specific block are changed accordingly.



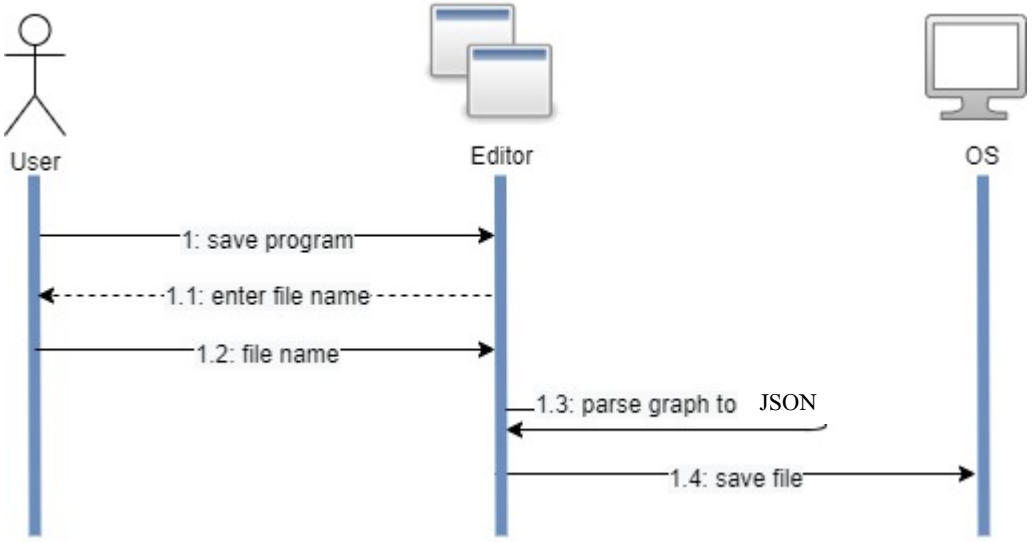
6.

Use-case name	Loading a program
Description	The user is pressing the load-program button, picks a JSON file and opens it.
Pre-conditions	None.
Post-conditions	The working-sheet filled with the graph represented by the structure inside the JSON file.



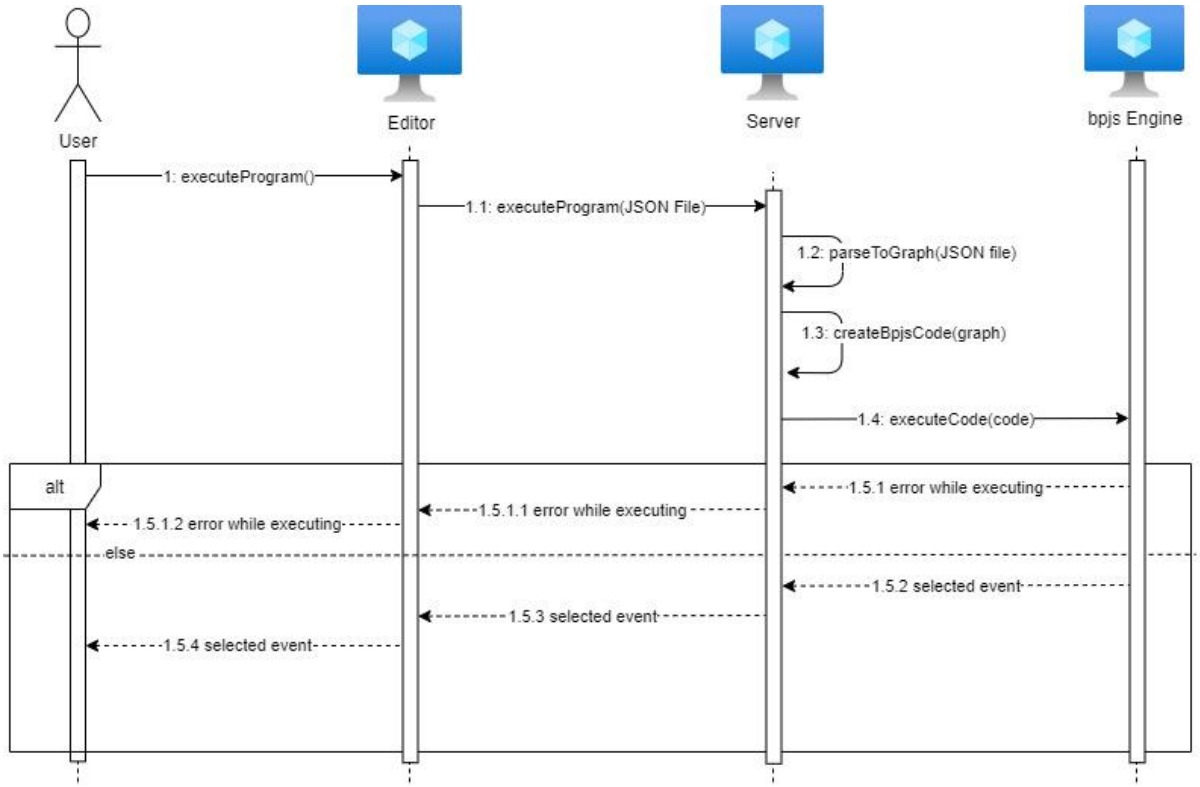
7.

Use-case name	Saving a program
Description	The user is pressing the save-program button, picks location and file name and saves it.
Pre-conditions	None
Post-conditions	An JSON file created on the selected location which contains all the graph details.



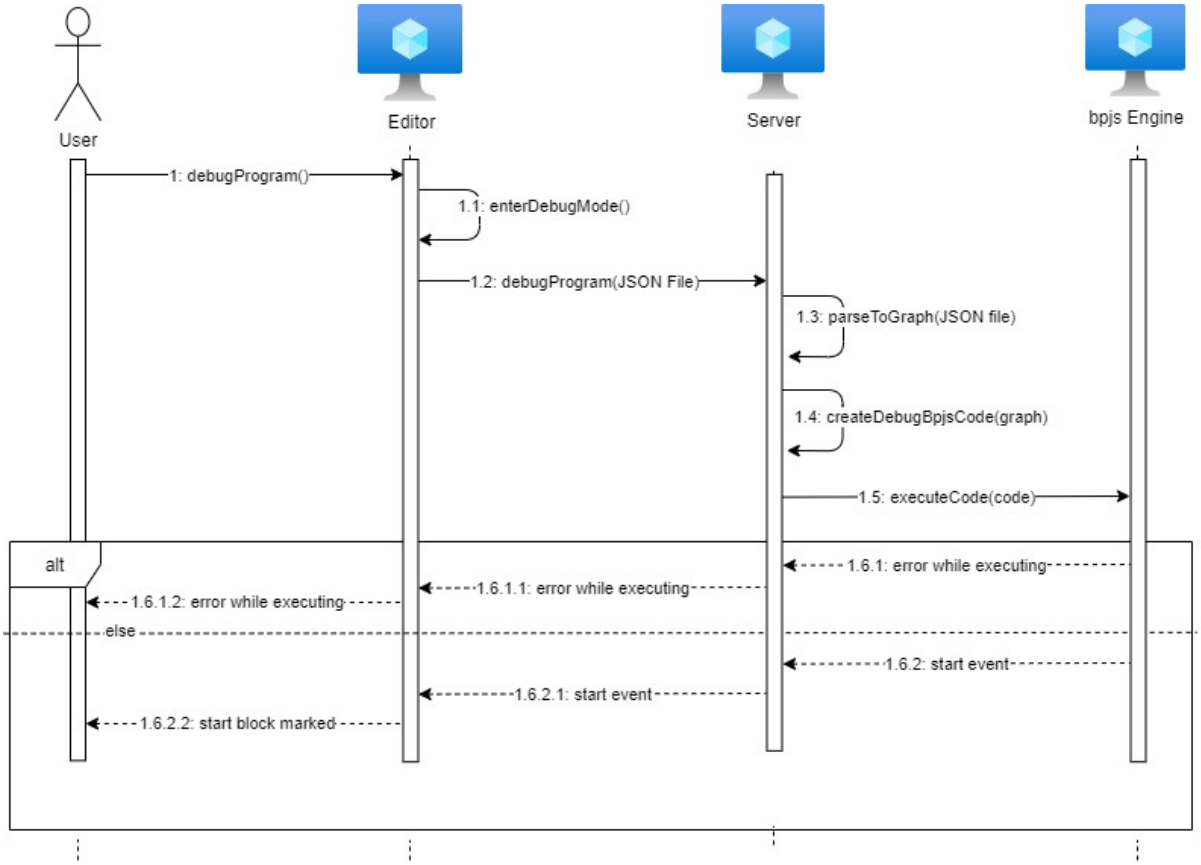
8.

Use-case name	Executing a program
Description	Pressing the run button after a graphical BP Flow program is located on the working sheet. A JSON presents the graph being sent to the server. The server parses and converts the JSON object to a graph-like object. The server creates a bpjs code file based on the graph structure. The server executes the bpjs code using bpjs server.
Pre-conditions	Loaded/Created graph bases on the working-sheet.
Post-conditions	The output console filled with events occurred according to the program restrictions.
Alternatives:	One of the general blocks' code section has code that breaks in run-time. The execution will stop, and an appropriate message will be shown to the user.



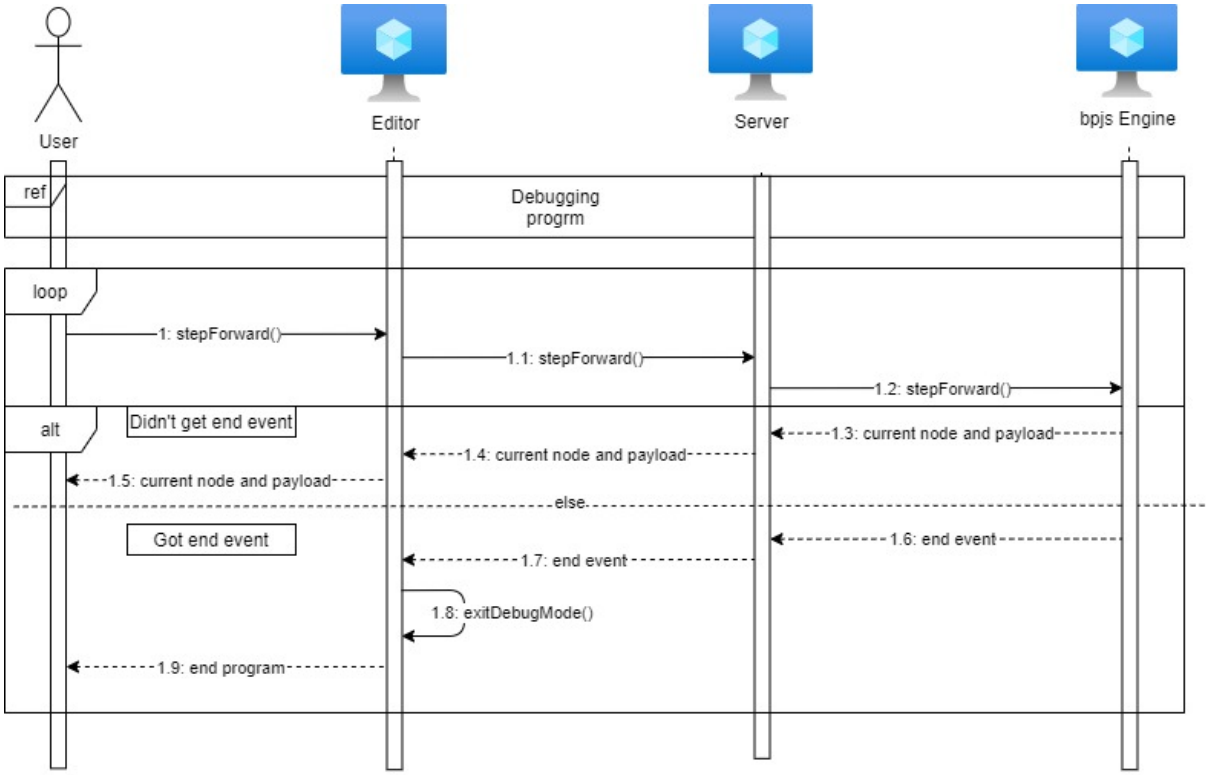
9.

Use-case name	Debugging a BP-Flow program
Description	After a graphical program is located on the working sheet, pressing the debugging button will transform the UI into debug mode. A JSON presents the graph being sent to the server. The server parses and converts the JSON object to a graph-like object. The server creates a **debug** bpjs code file based on the graph structure. The server executes the bpjs code using bpjs server.
Pre-conditions	Loaded/Created graph bases on the working-sheet.
Post-conditions	Step Forward and stop buttons are available, action that changes the program semantics are blocked until exiting debug mode. Start block of the executing flow is marked.
Alternatives:	There was an error while executing the code. an appropriate message will be shown to the user



9.1

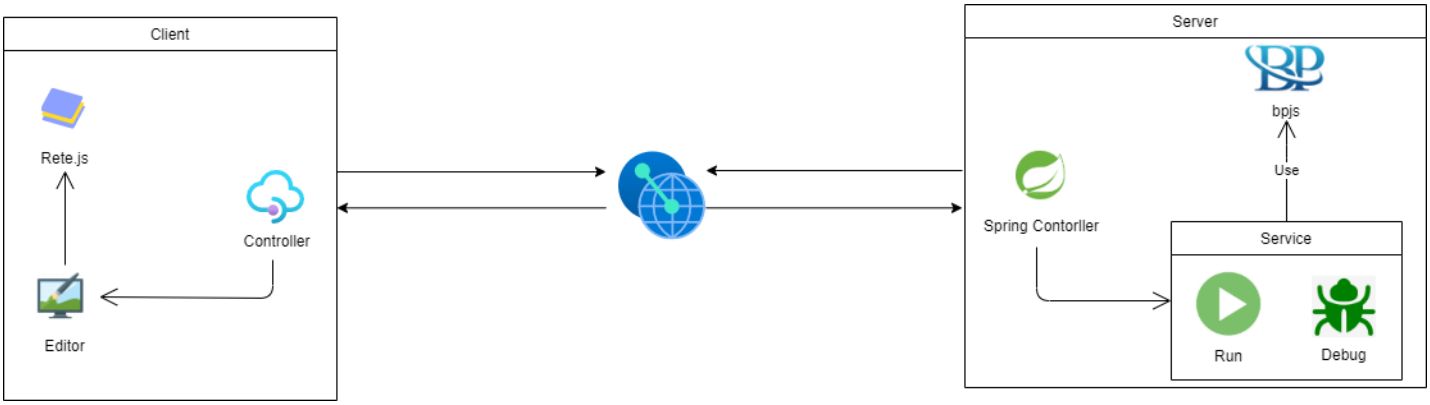
Use-case name	Step forward in debug mode.
Description	While in debug mode, the user can click on the step – forward button, to see the next step of the program that lies on the working sheet.
Pre-conditions	Debug button was clicked, and the editor is in debug mode.
Post-conditions	The next step of the program execution will be visible to the user.
Alternative	The bpflow program has reached the end of the execution, and therefore editor exits from debug mode.



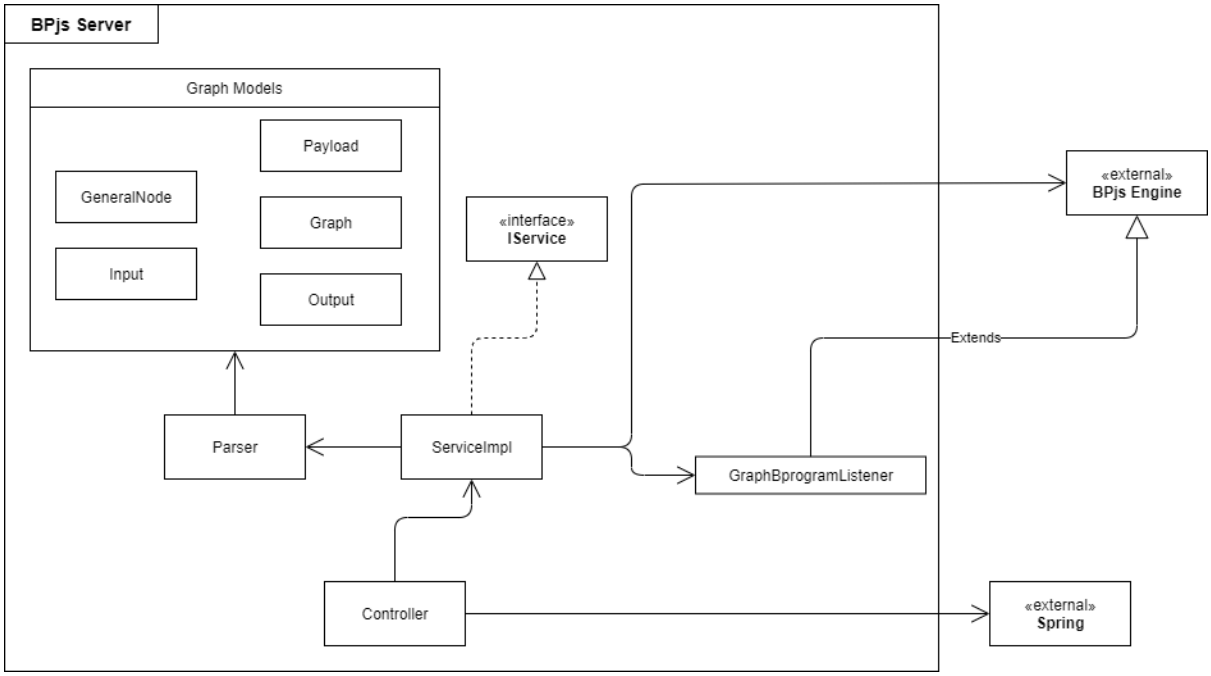
Chapter 2

System Architecture

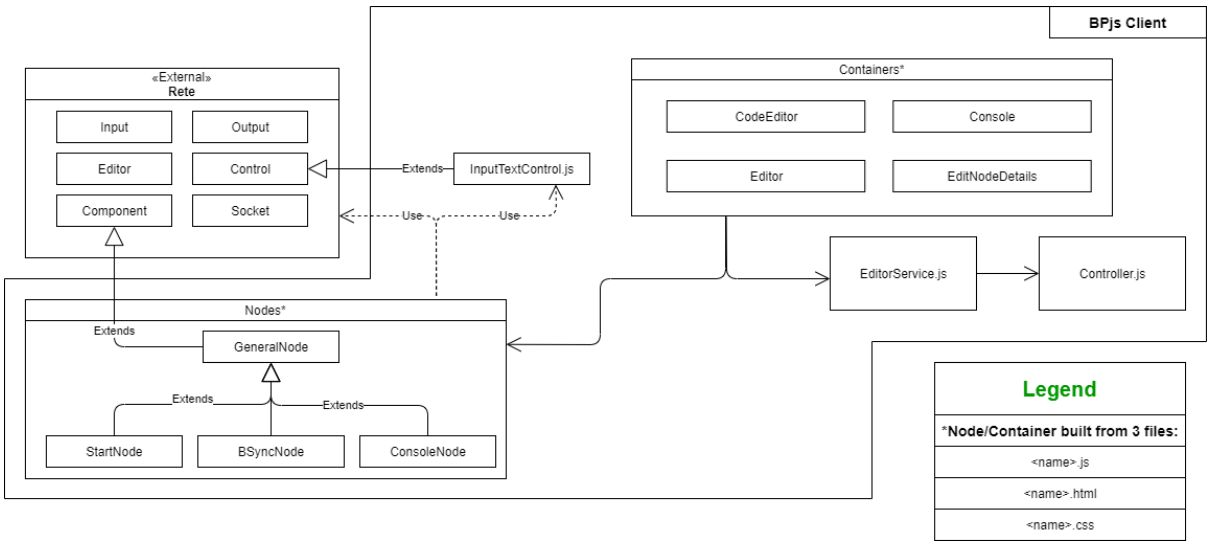
The system client is based on the Rete.js graphical editor library.
The system server based on Spring as communication layer, and uses bpjs server for the bp program execution.



Server Class Diagram



Client Class Diagram



Chapter 3 – Tests

Non-functional requirements:

- Visualization and portability requirements will be tested manually.

Functional requirements:

- Client-side: using Selenium tool.
- Server-side: using JUnits.

1. BP Flow program creation Tests – Using selenium

Req No.	Test description	Input	Expected
1.1	Create a new blank working sheet.	Choose blank working sheet	The app opens correctly without exceptions, and the graph object Is empty.
1.2	Create a new working sheet from default examples.	Choose example working sheet	The app opens correctly the chosen example, and the graph object is compatible.
1.3.1	Save a working sheet to the user's working space.	Clicking the save button, clicking “save to device” button file name	File including the diagram being saved in the user’s device in the selected location in the user’s device.
1.3.2	Save a working sheet to the user's drive.	Clicking the save button, clicking “save to drive” button, file name, drive details.	File including the diagram being saved in the user’s device in the matching drive.
1.4.1	Load a saved working sheet from the user’s working space.	**Good:** Clicking on “open” button, Clicking “open from device” button, a valid graph file.	The app opens correctly the chosen graph file, and the graph object is compatible.
		Bad: Clicking on “open” button, Clicking “open from device” button, an invalid graph file.	The app displays an error message, and the graph object is empty.

1.4.2	Load a saved working sheet from the user's drive.	**Good:** Clicking on "open" button, Clicking "open from drive" button, a valid graph file.	The app opens correctly the chosen graph file, and the graph object is compatible.
		Bad: Clicking on "open" button, Clicking "open from drive" button, an invalid graph file.	The app displays an error message, and the graph object is empty.
1.5	Add a free-text box to the working sheet.	Clicking on the right mouse button, choose "text box".	The text box appears in the working sheet, and the graph object is compatible.
1.6.1	Provide a tool set that includes the following objects: • General transformation node • Start node • B-Sync node • Console node.	Clicking on the right mouse button.	The list of the following nodes is displayed: • General transformation node • Start node • B-Sync node • Console node.
1.6.2	Define initial payloads in a start node.	Clicking on the Start node, insert payload.	Payload is visible in the text area designated to it, , and the graph object is compatible.
1.6.3.1	Define a title for a General node.	Clicking on a General node, insert title.	The inserted node title is displayed on the top of the General node, and the graph object is compatible.
1.6.3.2	Define the number of outputs for a General node.	**Good:** Clicking on a General node, insert a non-negative number of outputs.	The number of the General node outputs is updated to be the inserted number, and the graph object is compatible.
		Bad: Clicking on a General node, insert a negative number of outputs.	The app displays an error message, and the graph object isn't changed.

1.6.4	Writing code for a General node.	**Good:** Clicking on a General node, insert code without syntax errors.	The inserted code saved in the General node, and the graph object is compatible.
		Bad: Clicking on a General node, insert code with syntax errors.	The app displays an error message, and the graph object isn't changed.
1.6.5	Define requested, waited-for and blocked events on a Bsync node.	Insert a text in the corresponding text boxes.	The inserted text is saved and shown, and the graph object is compatible.
1.7	Create the nodes from pop-up menu.	Clicking on the right mouse button, choose the wanted node type.	The chosen node is displayed on the working sheet, and the graph object is compatible.
1.8	Drag the nodes from the tool set and drop them on the working sheet.	Drag the chosen nodes to the working sheet.	The selected node on the working sheet, and the graph object is compatible.
1.9	Move nodes from one place to another on the working sheet.	Drag the node from one place to another.	The node is displayed in the new location, and the graph object is compatible.
1.10	Delete node from the working sheet.	Right click on mouse button on the node, select delete.	The node isn’t displayed, and the graph object is compatible.
1.11	Connect nodes by arrows.	**Good:** Click on the right circle on the node, drag the mouse to the left circle of another node.	An arrow between the two nodes is displayed, and the graph object is compatible.
		Bad: Click on the right circle on the node, drag the mouse to the right circle of another node.	Nothing is changed on the working sheet, and the graph object isn't changed.
		Bad: Click on the left circle on the node, drag the	

		mouse to the left circle of another node.	
		Bad: Click on the left circle on the node, drag the mouse to the right circle of another node.	
1.12	Clone an existing node.	Right click on mouse button on the node, select clone.	The selected node will be displayed twice on the working sheet, and the graph object is compatible.

2. BP Flow program visualization Tests:

Req No.	Test description	Input	Expected
2.1	Insert a node between two existing nodes connected by an arrow.	Drag a node between two connected nodes, place it on the arrow between them.	The dragged node will be connected between the two existing nodes, and the graph object is compatible.
2.2	Moving existing nodes during debug mode execution, as long as the changes do not affect the program's semantic.	**Good:** Enter debug mode, drag a node to a different location on the working sheet.	The node will be placed in the new location, and the graph object is compatible.
		Bad: Enter debug mode, delete an arrow.	The action will be blocked, nothing changed on the working sheet, and the graph object isn't changed.
		Bad: Enter debug mode, add a new arrow between two nodes.	
		Bad: Enter debug mode, delete a node.	

2.3.1	Display the payload of each node during each step in a step-by-step execution.	Create a bp flow graph, click on the debug button, click “step”.	The payload received in each node after the first step will be displayed.
2.3.2	Mark the current running node.	Create a bp flow graph, click on the debug button, click “step”.	The last node that was selected after the first step is marked.

Test integration:

All the following test of execution and debug will be tested in the way explained below:

- Using real server and mock client.
- Using mock server and real client.
- Using real server and real client (E2E).

3. Execution:

Description	Goal	Input	Expected Result
Hello World: Checks the order of event requests that occur.	Legal order of events occurs from one scenario.	Json that represents a diagram of BP Flow syntax that represents “hello world” program.	Two events in the following order: “Hello”, “World”.
Random Order: Checks that the probability of choosing to execute the same sequence of requests in 100 runs is between 45% to 55% (from two different requests sequence)	Choosing events sequences from uniform distribution.	Json that represent a diagram of BP Flow syntax built of two different sequences.	A sequence of 2 events: "1", "2" or "2", "1"

Hot Cold: Checks the program Hot Cold - Checks the order of event requests that occur in conjunction with block and wait events	sequence of events occurs from more than one scenario that include: Request, block and wait events in bsync nodes.	Json that represent a diagram of BP Flow syntax	Legal order of events: Hot, Cold, Hot, Cold, Hot, Cold
Payload: Checks that the payloads that are inserted in the start node pass between nodes, checks the value of them.	Current Payloads pass between nodes.	Json that represent a diagram of BP Flow syntax	The correct value of the payloads: [{"x":3}, {"y":4}]
Payload Change: Checks that the value of the payloads that are inserted in the start node could be changed in a general block. passes the payloads with the new changes between nodes and checks the current new value of them.	Payloads can change in general node.	Json that represent a diagram of BP Flow syntax	The correct value of the payloads after changes of their values: [{"x":5}]
Payloads If Else: Checks that general node send other payloads to other outputs, according to the user-defined in the "if-else" condition which is found in the code editor of the general node.	General node pass payload to selected exit outputs point.	Json that represent a diagram of BP Flow syntax	The correct value of the payloads that pass from different outputs: {"x":3}
Illegal Graph: checks if the graph has a disconnected start node.	Detection of illegal graph elements.	Json that represent a diagram of BP Flow syntax, that include a disconnected start node.	The app displays an error message.
Exception Handle: checks that when error occurs while executing the JS code of a general node, the execution of the scenario is terminated.	handle error while executing JS code.	Json that represent a diagram of BP Flow syntax, that includes JS code in the code editor of the general node that raised an exception.	The app displays an error message.

Tic-Tac-Toe: Checks the complex program Tic-tac-toe. - Checks the order of events, the amount of events and the rules of the game	Legal order of events occurs, General-node multiply outputs And Request, Block and Wait events in bsync nodes	Json that represent a diagram of BP Flow syntax	Number of events that selected :9. Legal order of occurrences: “X”, ”O”, ”X”, ”O” … Valid selection of game board slot

4.Debugging:

Description	Goal	Input	Expected Result
Hello World: Checks the order of event requests that occur.	Legal order of events occurs from one scenario.	Json that represents a diagram of BP Flow syntax that represents “hello world” program.	Two events in the following order: “Hello”, “World”.
Random Order: Checks that the probability of choosing to execute the same sequence of requests in 100 runs is between 45% to 55% (from two different requests sequence)	Choosing events sequences from uniform distribution.	Json that represent a diagram of BP Flow syntax built of two different sequences.	A sequence of 2 events: "1", "2" or "2", "1"
Hot Cold: Checks the program Hot Cold - Checks the order of event requests that occur in conjunction with block and wait events	sequence of events occurs from more than one scenario that include: Request, block and wait events in bsync nodes .	Json that represent a diagram of BP Flow syntax	Legal order of events: Hot, Cold, Hot, Cold, Hot, Cold

Payload: Checks that the payloads that are inserted in the start node pass between nodes, checks the value of them.	Current Payloads pass between nodes.	Json that represent a diagram of BP Flow syntax	The correct value of the payloads: [{"x":3}, {"y":4}]
Payload Change: Checks that the value of the payloads that are inserted in the start node could be changed in a general block. passes the payloads with the new changes between nodes and checks the current new value of them.	Payloads can change in general node.	Json that represent a diagram of BP Flow syntax	The correct value of the payloads after changes of their values: [{"x":5}]
Payloads If Else: Checks that general node send other payloads to other outputs, according to the user-defined in the "if-else" condition which is found in the code editor of the general node.	General node pass payload to selected exit outputs point.	Json that represent a diagram of BP Flow syntax	The correct value of the payloads that pass from different outputs: {"x":3}
Illegal Graph: checks if the graph has a disconnected start node.	Detection of illegal graph elements.	Json that represent a diagram of BP Flow syntax, that include a disconnected start node.	The app displays an error message.
Exception Handle: checks that when error occurs while executing the JS code of a general node, the execution of the scenario is terminated.	handle error while executing JS code.	Json that represent a diagram of BP Flow syntax, that includes JS code in the code editor of the general node that raised an exception.	The app displays an error message.
Tic-Tac-Toe: Checks the complex program Tic-tac-toe. - Checks the order of events, the amount of events and the rules of the game	Legal order of events occurs, General-node multiply outputs And Request, Block and Wait events in bsync nodes	Json that represent a diagram of BP Flow syntax	Number of events that selected :9. Legal order of occurrences: “X”, ”O”, ”X”, ”O” … Valid selection of game board slot