Flow BP

Software Engineering Project

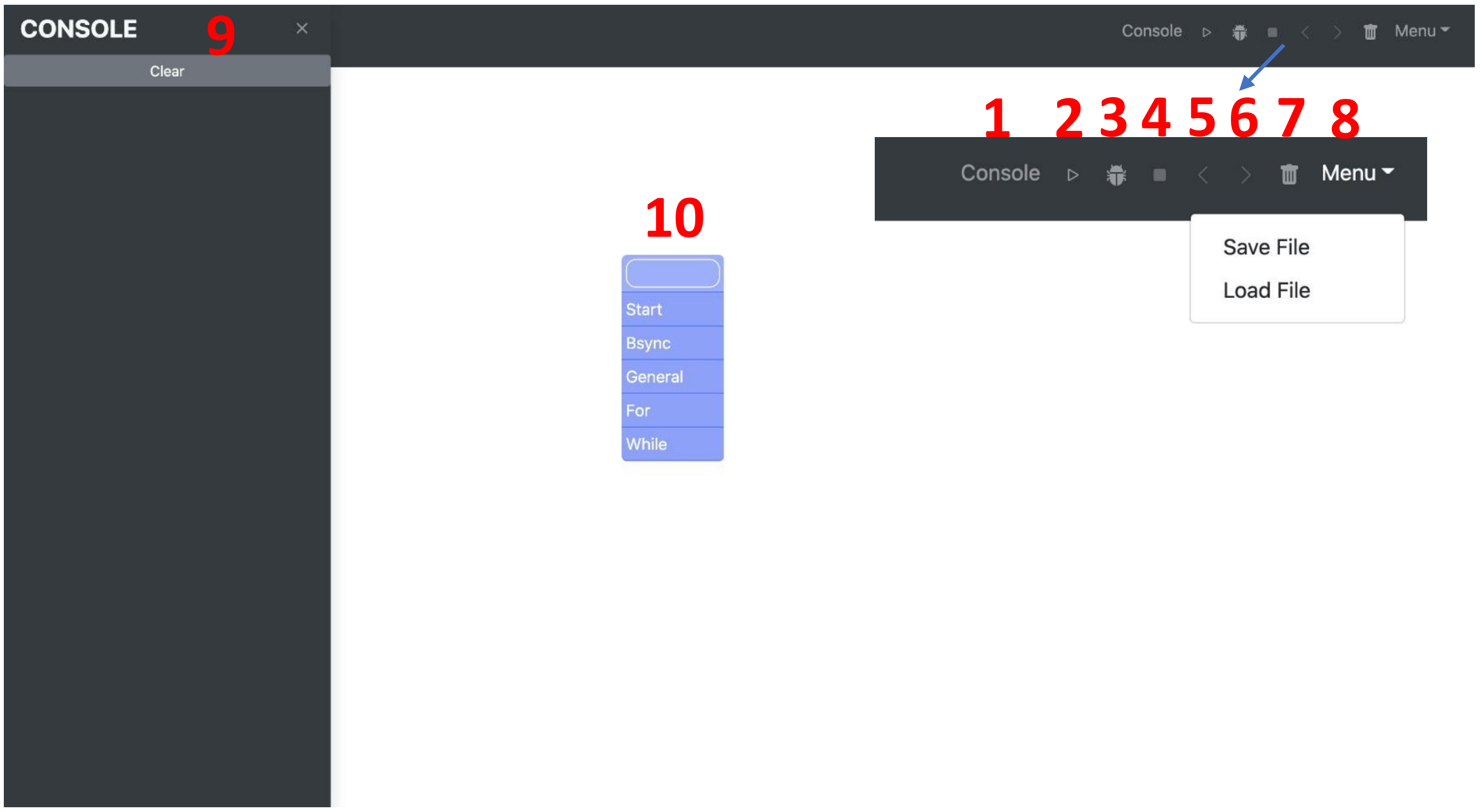
User Manual

Group 4

Tomer Bitran
Shir Markovits
Shahar Hazan

Contents

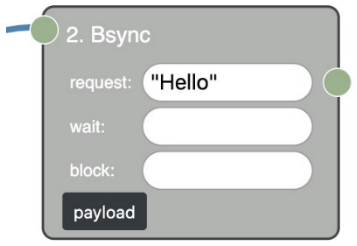
Opening screen



Legend

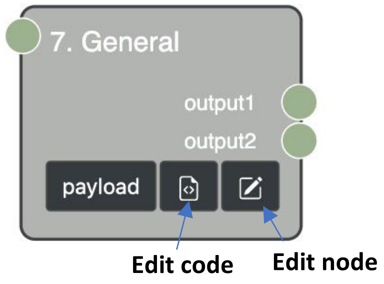
1. Console: opens a console bar in the left side of the screen showing the events where running after running the “Run”/”Debug” (#2/#3) buttons
2. Run: after creating a flow, runs the BP program derived from the created flow.
3. Debug: after creating a flow, turns the program into “debug” mode, disables all other buttons except “stop”, “step back” and “step forward” (#4, #5, #6).
4. Stop: turn off the debug mode. Enabled only while in debug mode.
5. Step back: not implemented.
6. Step forward: enabled only while in debug mode. Show the step just happened in the created flow: every first click paints in green the selected nodes that run, and every second click paints in dark gray the active nodes that are possible to be selected in the next click.
7. Trash: deletes the flow already created and remains a clean working sheet.
8. Menu: enables the user to load a ready-made flow and save the current flow.
9. Clear: clears the presented output on the console bar.
10. The nodes menu:
    a. Start: a node presents a new BThread. The user can initialize the payload will be transferred to the next node.

b. BSync: a node that presents a bsync object. The user can assign an even the node will be waiting for, an event the node will block and an event to be happen when the node will be selected.

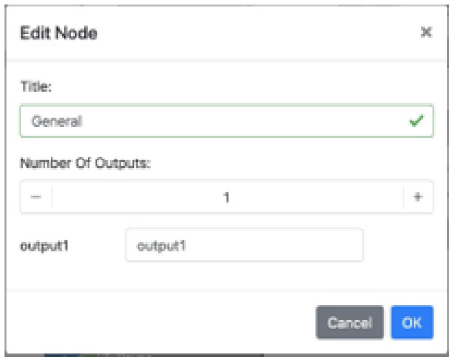


   i. Those fields can get:
      1. a **string** like: “Hello”
      2. a **json** object like: {name: <event name> data: js-type}
      3. a **list of jsons** like: [{name: <event name> data: js-type}, {name: <event name> data: js-type}]

c. General: a node type can be placed between two nodes. This node can contain a javascript code and several outputs.



   i. Editing a General node:
      1. The user can modify the node’s title, the number of outputs it will have and their names.



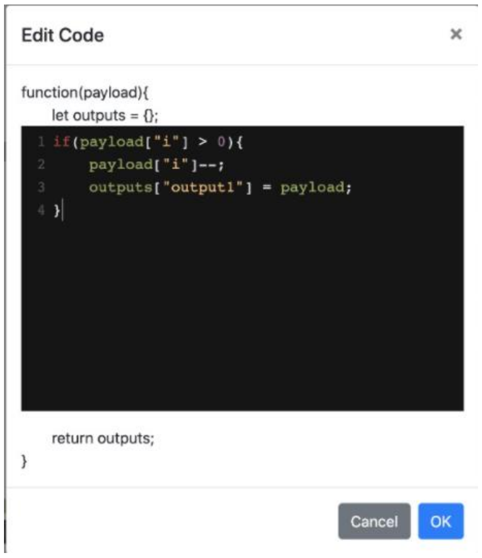
Title: General

Number Of Outputs: 1

output1: output1

ii. Editing General node's code:

   1. **Note** – in the end of the javascript code the user must define the payload to pass for each output assign to the node. It should be written this way:
      Outputs[<output name>] = <payload to pass>

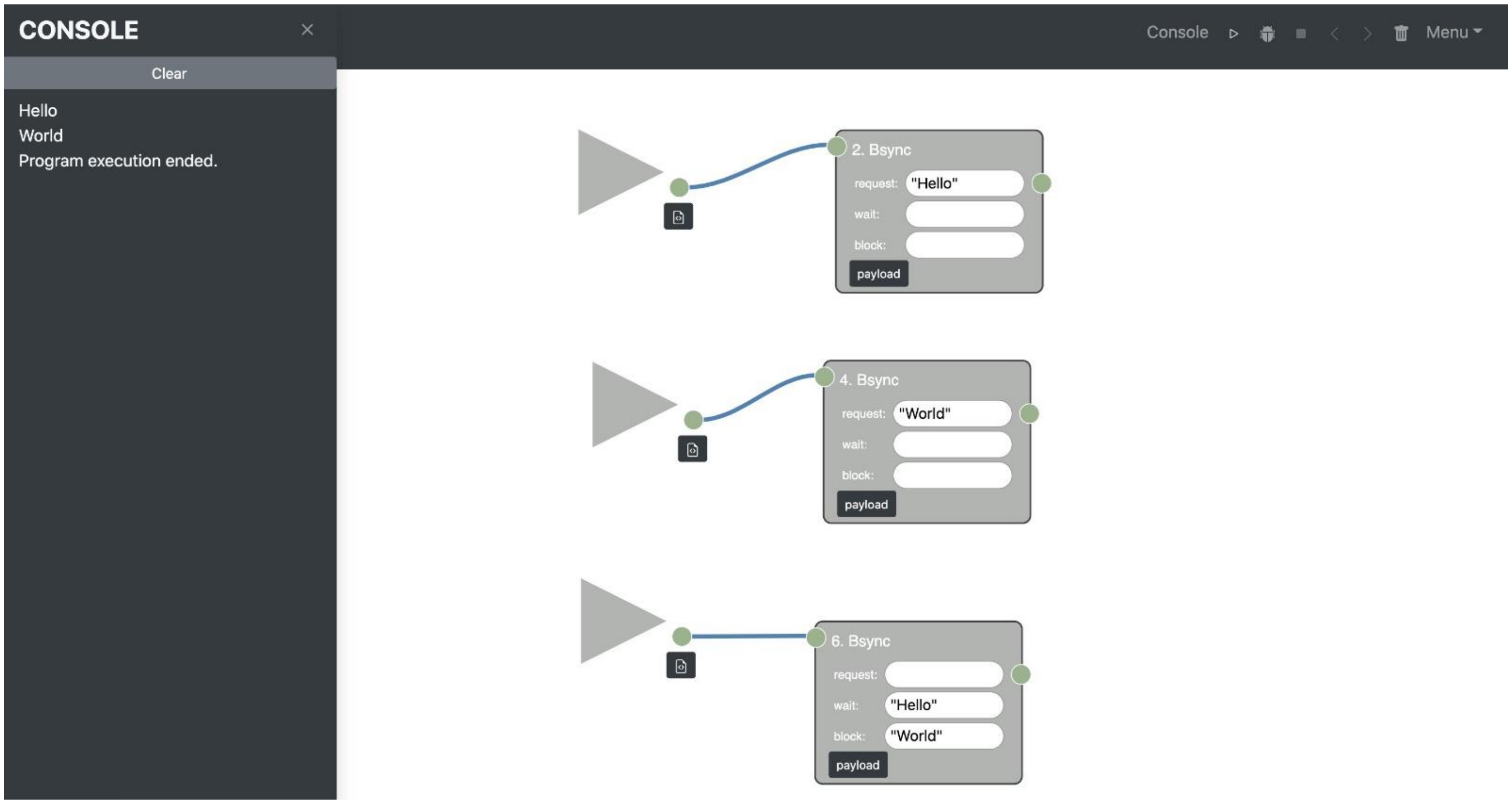


```
function(payload){
    let outputs = {};
if(payload["i"] > 0){
    payload["i"]--;
    outputs["output1"] = payload;
}

    return outputs;
}
```

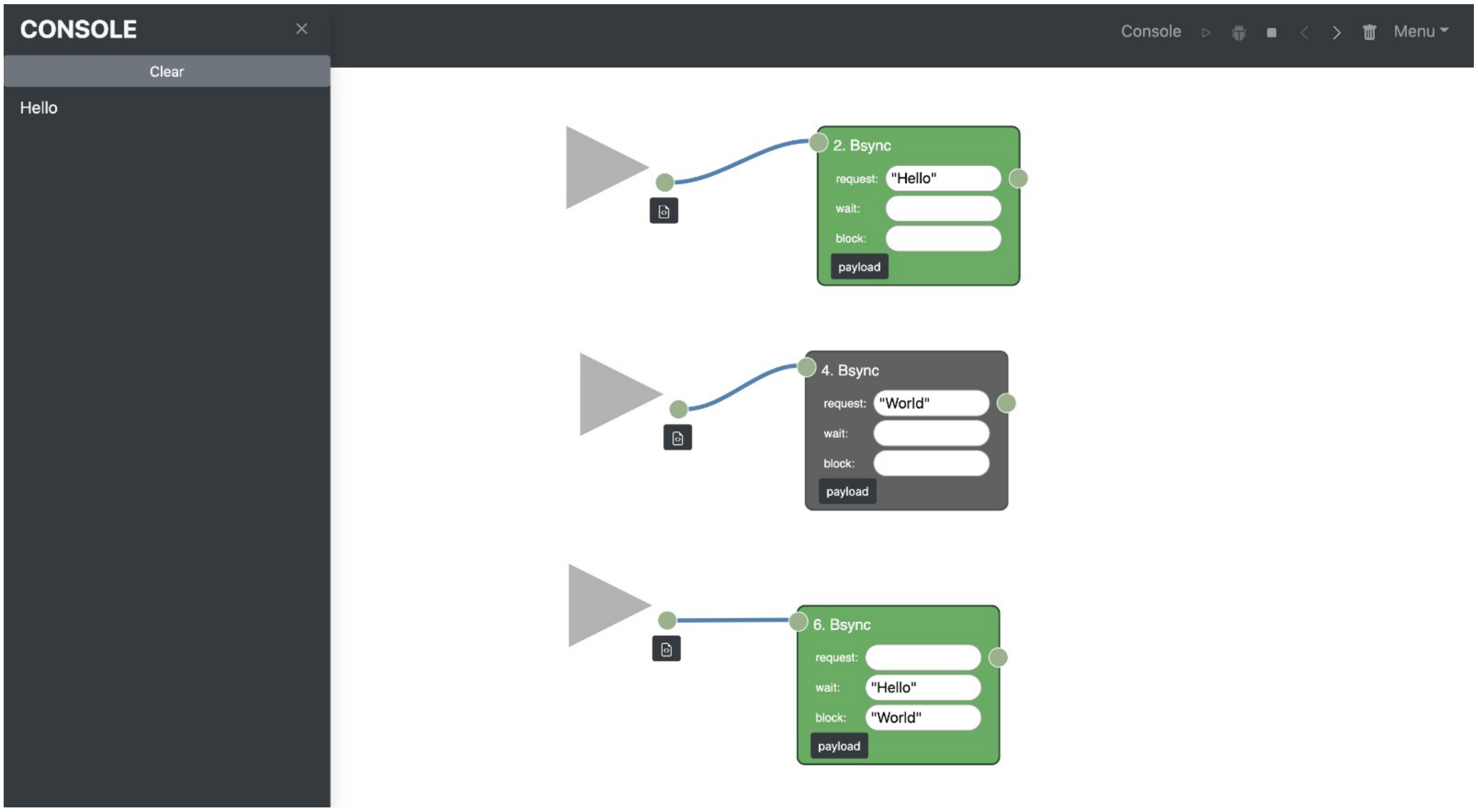
Creating “Hello World” flow

1. Right click on the screen presents the nodes menu.
2. Create a start node to be the initializer of “Hello” event.
3. Create another start node to be the initializer for the “World” event.
4. Create another start node to be the initializer for the “arbiter” thread.
5. Create a bsync node to handle the “Hello” event.
6. Create a bsync node to handle the “World” event.
7. Create a bsync node to be responsible for waiting for “Hello” and blocking “World”.
8. Connect each start node to its relevant scenario.
9. For presenting the output of the created program click on “Console” button.
10. Click run: the events will be shown on the Console bar.



```
Hello
World
Program execution ended.
```

11. For debugging:
    a. Click “Debug” to turn into debug mode. The possible nodes to be selected will be shown in dark gray.
    b. Click “step forward”: the selected nodes will be shown in green.
    c. To turn off the debug mode click stop, to continue the debugging keep clicking “step forward” until program will be ended.



Hello

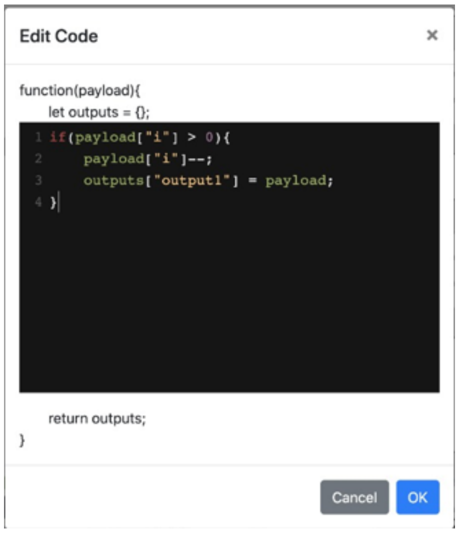
"Hello"

"World"

"Hello"
"World"

Creating “Generic Hot Cold” flow

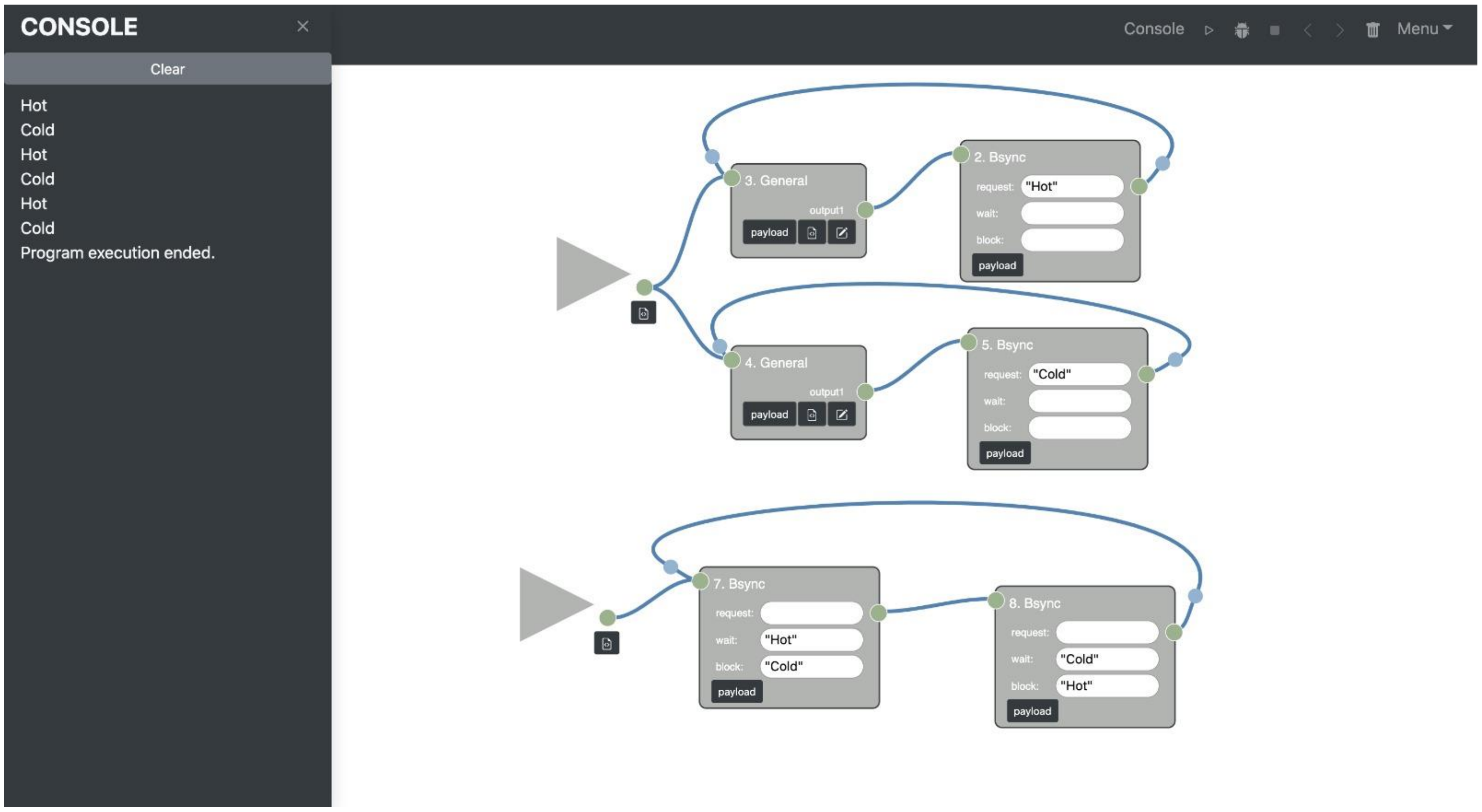
1. Create a start node to be the initializer of both “Hot” and “Cold” events.
    a. Initialize the payload it passes to the connected events this way: {“i” : 3} (in this case we want each event, “Hot” and “Cold” to happen three times.
2. Create another start node to be the initializer for the “arbiter” thread.
3. Create a bsync node to handle the “Hot” event.
4. Create a bsync node to handle the “Cold” event.
5. Create two general nodes (from the two declared above) to handle the code that will make sure each one (“Hot”, “Cold”) will happen exactly three times. (in this case each node will decrement by 1 the counter each time the related event is happening).



```
function(payload){
    let outputs = {};
if(payload["i"] > 0){
    payload["i"]--;
    outputs["output1"] = payload;
}
    return outputs;
}
```

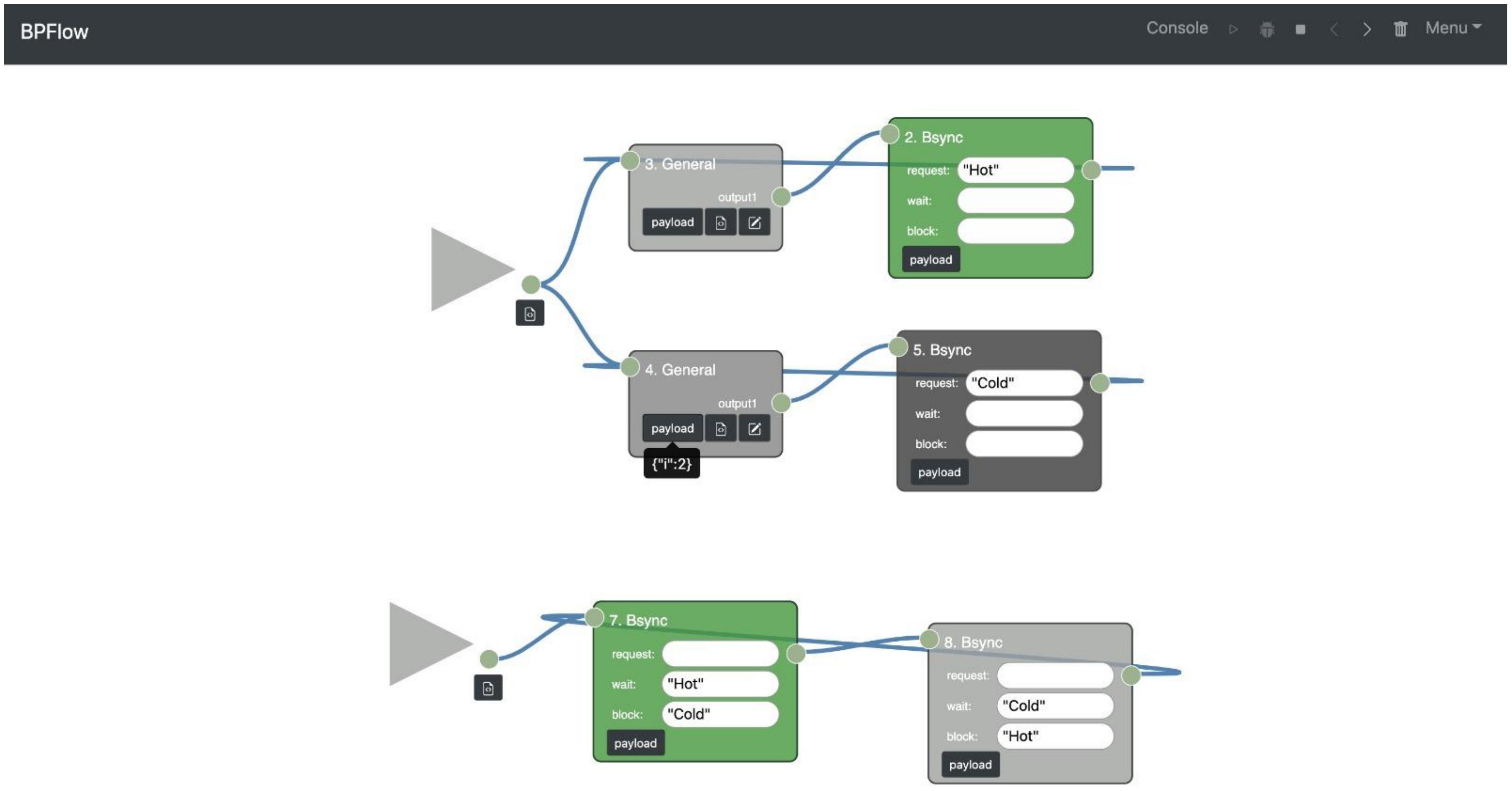
6. Add two bsync nodes to be the arbiters that make sure “Hot” and “Cold” will happen alternately -> first bsync node waits for “Hot” and blocks “Cold” and the second one will wait for “Cold” and will block “Hot”.
7. Connect each start node to its relevant scenario.
8. For presenting the output of the created program click on “Console” button.

9. Click run: the events will be shown on the Console bar.



```
Hot
Cold
Hot
Cold
Hot
Cold
Program execution ended.
```

10. For debugging:
    a. Click “Debug” to turn into debug mode. The possible nodes to be selected will be shown in dark gray.
    b. Click “step forward”: the selected nodes will be shown in green.
    c. To turn off the debug mode click stop, to continue the debugging keep clicking “step forward” until program will be ended.


BPFlow
Console
Menu
3. General
output1
payload
4. General
output1
payload
{"i":2}
2. Bsync
request: "Hot"
wait:
block:
payload
5. Bsync
request: "Cold"
wait:
block:
payload
7. Bsync
request:
wait: "Hot"
block: "Cold"
payload
8. Bsync
request:
wait: "Cold"
block: "Hot"
payload