Synchronous vs. Asynchronous JavaScript

Table of Contents

Outline **2**
Resources 2
Scenario 2
How-To **3**
Getting Started 3
1. Set up an Asynchrounous Client Action 3
Extra task! 4
2. Build the game page 5
3. Execute the Asynchronous action 6
Extra task! 7

knowledge@outsystems.com

www.outsystems.com

© 2022 by OutSystems. All rights reserved. OutSystems is a registered trademark of OutSystems. The information contained herein is subject to change without notice. All terms and conditions for OutSystems products and services are set forth in the agreements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. OutSystems shall not be liable for technical or editorial errors or omissions contained herein.

Outline

In this exercise we will focus on building actions to work on a asynchronous way. Upon completion, we'll have built a page with a simple with that emulates a number guessing game. Then on the application page itself we will use the browser console to analyse the promise queue.

Resources

This exercise can be set up on a new application. For further mention on this exercise it is implicit to have a completely blank Reactive Web App with one single UI module.

Scenario

In this exercise, we'll start to create the new Reactive Web application as well a single module for UI purposes. **JavascriptFrontend** will be our app name plus the Reactive web Module named **JavascriptASyncDemo**.

How-To

In this section, we'll show you how to do this exercise, with a thorough step-by-step description. **If you already finished the exercise on your own, great! You don't need to do it again.** If you didn't finish the exercise, that's fine! We are here to help you.

Getting Started

To start this exercise, we need first to create a new Reactive Web Application from scracth and name it **JavascriptFrontend**. For the time being you are free to choose any color in the app name and description form or optionally upload a custom icon. Next, please start to create a new Reactive Web Module type and name it **JavascriptASyncDemo** and we are done.

1. Set up an Asynchrounous Client Action

At the Logic Tab we start to build an asynchrounous action that will be called from the screen action.

1) Create a new Client Action ***ASynchronousGuess*** with the following parameters:

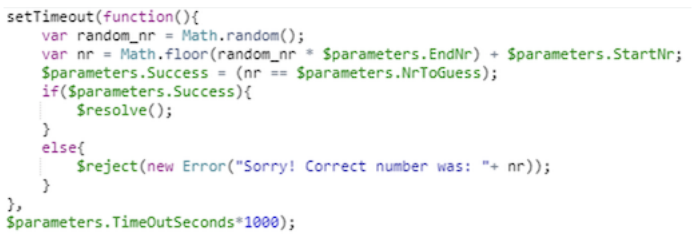
    - an input parameter : *InGuessNumber*, Integer, Mandatory
    - an output parameter: *Success*, Boolean, Default Value as *False*

2) At action flow, drag a Javascript node after the Start Node and name it **AsyncGuess**

3) Open the **AsyncGuess** Javascript node and set the following parameters:

    - an input parameter : *NrToGuess*, Integer, Mandatory
    - an input parameter : *TimeOutSeconds*, Integer, Mandatory
    - an input parameter : *StartNr*, Integer, Mandatory
    - an input parameter : *EndNr*, Integer, Mandatory
    - an output parameter: *Success*, Boolean, Default Value as *False*

4) Inside the code editor place the Javascript code:



```
setTimeout(function(){
    var random_nr = Math.random();
    var nr = Math.floor(random_nr * $parameters.EndNr) + $parameters.StartNr;
    $parameters.Success = (nr == $parameters.NrToGuess);
    if($parameters.Success){
        $resolve();
    }
    else{
        $reject(new Error("Sorry! Correct number was: "+ nr));
    }
},
$parameters.TimeOutSeconds*1000);
```

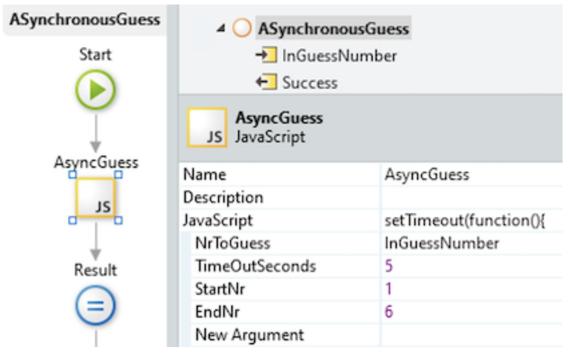
1) Close the editor and click again on the **AsyncGuess** Javascript node and set the input parameters as:

- *NrToGuess* = *InGuessNumber* (action parameter)
- *TimeOutSeconds* = 5
- *StartNr* = 1
- *EndNr* = 6

2) After the **AsyncGuess** element set a new Assign node to the client action output parameter:

- Succes = **AsyncGuess.Success**

After you complete these steps the flow should be as the image below.



Name	AsyncGuess
Description	
JavaScript	setTimeout(function(){
NrToGuess	InGuessNumber
TimeOutSeconds	5
StartNr	1
EndNr	6
New Argument	

Extra task!

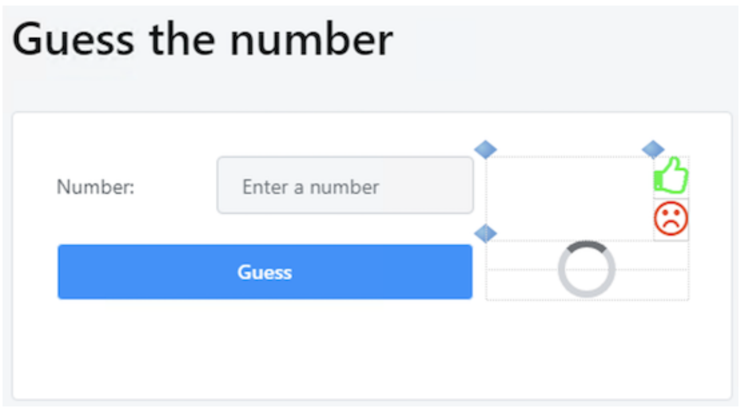
Build auxiliar Client Variables and actions so the input values to the Javascript node are set programatically in the flow. Tip: create a new structure called **GuessConfiguration**

Publish your changes.

2. Build the game page

Before we start on doing asynchronous calls, we need to first build a screen for our litle guessing game. When the user guesses the random number generated by the asynchrouns action the screen will produce a success or failure notification to the user.

In the end we will try to have a screen built like the image below. (canvas preview).



1) At the Interface tab, create a new Screen named **GuessTheNumber**

2) In the new screen please create:

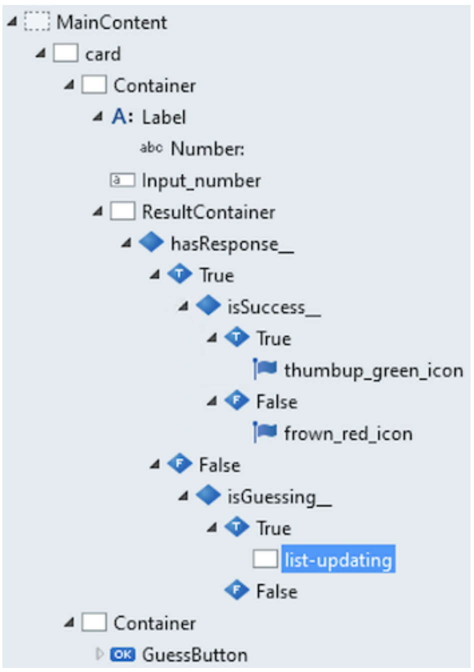
   - three new local variables: *Boolean* datatype; Default value to *False*.
     - **hasResponse**, **isGuessingStarted**, and **isSuccess**
   - an additional local variable called **Number**, Integer
   - a new Screen Action named **Guess**

Do the screen building process starting by the Button, Input Field and a Label widgets. For the result side of the screen use Icon and If Widgets to set a small animation process while the Promise action is not yet fullfilled due to the setTimeout of the **ASynchronousGuess** asynchronous client action.

At the end the screen Widget Tree will be the following.

- Each **If** widget is named after the local screen Boolean variable(s).

- The input field widget variable is the **Number** local screen variable.



Publish your changes and open the **GuessTheNumber** page in the browser.

3. Execute the Asynchronous action

The **Guess** screen action will run the **ASynchronousGuess** asynchronous client action created before.

At the **Guess** action flow:

1) After the Start node add a new assign to the **isGuessingStarted** = *True*

2) At action flow, after the Assign drag a Run Client Action to run the **ASynchronousGuess** defined before

3) Yet on this node, set the **InGuessNumber** parameter to be the **Number** screen variable

4) Name the node **ASynchronousGuess**

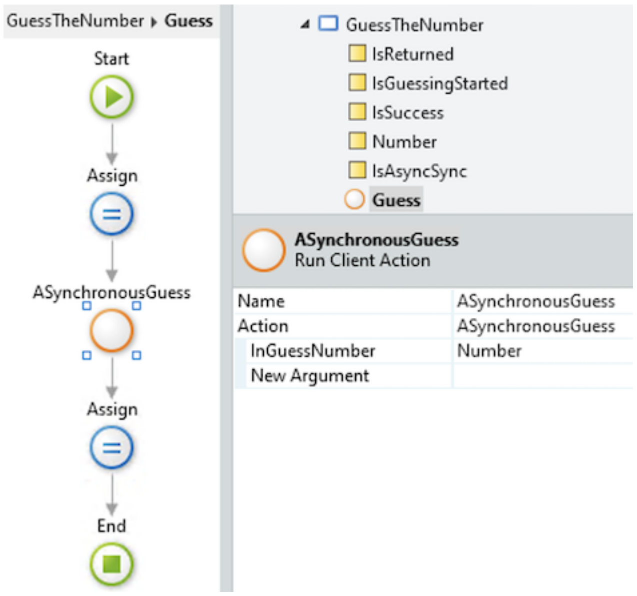
5) Drag a new Assign node afterwards, but before the End Node, with the assign statements:

    - **isSuccess** (screen local variable) = *ASynchronousGuess.Success*

    - **hasResponse** (screen local variable) = *True*

- **isGuessingStarted** (screen local variable) = *False*

After you complete these steps the flow should be as the image below.



Name	ASynchronousGuess
Action	ASynchronousGuess
InGuessNumber	Number
New Argument	

Publish your changes and open the **GuessTheNumber** page in the browser.

Afterwards, open the Console on the browser tools.

While testing some random numbers in order to guess, check what is happening in Console output.

Good luck!

Extra task!

Change whatever necessary to give back to the user screen which random number was the correct when it misses.