Filtering Inputted Values

Goal

1. **Only allow valid input values to be passed to the compute unit**
2. **Prevent hacker from purposefully entering invalid input values**

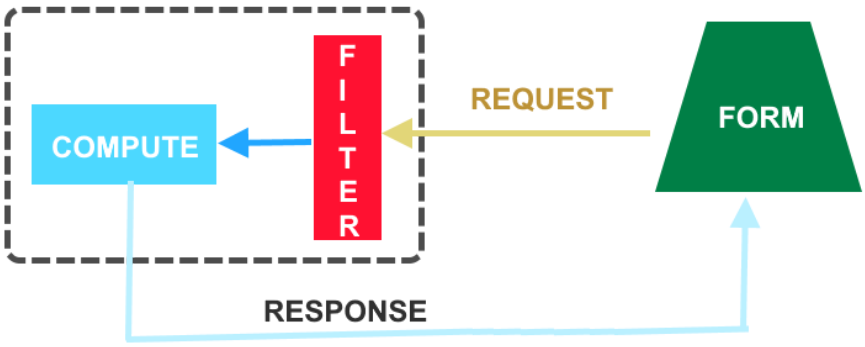
Context

Each user is unique and different from every other user of a device or service. The device or service must be able to handle each user's unique situation. How can this be accomplished? One way is to allow a user to customize the request that is sent to a server.

A form is one way to give users the capability to customize a request before it is sent to a server. Being able to customize a request allows a piece of code to be able to service specific cases instead of only a single generic case.

However, there is a problem with using a form to capture user input. The problem is that the user might accidentally or purposefully enter invalid input value, which is a value that the server is unable to handle.

Sometimes, the user might accidentally enter an invalid input value. However, there are situations where a hacker purposefully enters an invalid input value as a way to test the capability of the server.



Solution

The server should only accept and then process valid input values. How can the server do this?
One solution is to filter the input before passing it to the compute unit.
1. if the input value passes the filter, then it will be passed to the compute unit, which will use it to perform some type of computation by using the inputted value as an operand to generate a result.
2. however, if the input value does not pass the filter, then the input value is dropped

Assume that the server can only process decimal values. A filter must be created to make sure that only decimal values are entered as input values.

Implementation Of Solution

A filter can be implemented by using the "if" statement because an "if" statement is a decision block. Common values that could bring harm to the server are below. **These inputted values that could bring harm to the server become the input filter.**

1. blank input, ""
2. special characters,
3. underscore

Because there are 3 different types of values that bring harm to the server, this would require 3 “if” statements. The “if” statement will check if the user input is equal to any of the 3 types of data values.

Challenge, Bank Account Withdrawal Filter

Assume that your Bank Account currently has a balance of $100. You are an engineer who is tasked with building a withdrawal feature for a banking system. The user enters the amount of money to withdraw from the current balance. The values that could bring harm to the server are below. **Remember that the values that could bring harm to the server become the input filter.**

1. blank input, ""
2. special characters
3. a withdrawal amount that is greater than the amount of the balance

Super Challenge 1, Using Logical Operators

There are 3 filters and each filter requires the use of an "if" statement. For this Super Challenge, use a logical operator to combine all 3 "if" statements into a single "if" statement

This single "if" statement is able to filter out all input values that bring harm to the server

Super Challenge 2, Pop Up Helper Response

Sometimes, the user is not aware of the range of valid input values. What we can do is help the user by giving a pop up or response that displays the range of valid input values.

For this Super Challenge, have the pop up be inside the long “if” statement that was completed in Super Challenge 1.

A pop up can be displayed by using *alert(“Enter valid input”);* Everything inside the double quotes and open and closing parenthesis will be part of the pop up message to the user.