BONUS

Concept

CountOnMe est une application de calculette sur iphone. Elle est disponible en mode portrait.

Elle permet d'effectuer des opérations binaires, c'est à dire effectuer des opérations entre deux nombres.

Bonus

Utilisation de chiffre à virgule

Nous avons implémenté au modèle une fonction addDecimal() qui permet d'ajouter à la variable StringNumbers de type [String] un character "." en dernier character du tableau.

```
mutating func addDecimal(){
   if let stringNumber = stringNumbers.last {
      var stringNumberDecimal = stringNumber
      //Convert Int to String and append it to the former number
      stringNumberDecimal += "."
      // Replace formernumber with apended number
      stringNumbers[stringNumbers.count-1] = stringNumberDecimal
   }
 }
```

Derrière l'utilisateur peut uniquement inseré des chiffres car une fonction de validation empèche que StringNumbers contienne deux fois le caractère ".". Cette dernière renvoie un booleén.

```
  /**
   Check if the stack already contains a point or is empty and returns false it is
   */
  var canAddDecimal: Bool {
    if let strings = stringNumbers.last {
      if strings.contains(".") || strings.isEmpty {
        return false
      }
    }
    return true
  }
```

Cette fonctionnalité est implémentée dans le controller quand l'utilisateur tappe sur le bouton ".". Le controller appelle le model pour vérifier que l'utilisateur peut ajouter un point, puis si c'est possible ajoute le point.

```
    @IBAction func tappedPointButton(_ sender: Any) {
    if brain.canAddDecimal {
      brain.addDecimal()
      updateDisplay()
    } else {
      showAlert()
    }
  }
```

Dans le cas ou cela n'est pas possible une alerte est présenté par une alerte

L'utilisation de nombres décimaux au niveau du résultat est gèrée grace à une fonction de vérification et une fonction pour arrondir le résultat s'il s'agit d'un nombre entier.

Fonction de validation

Si le remainder d'un nombre diviser par lui même est égale à 0 alors il s'agit d'un nombre entier.

```
mutating func roundEvaluation(_ result: Double) -> Bool{
    if result.truncatingRemainder(dividingBy: 1) == 0 {
      return true
    }
    return false
  }
```

Dans le controller elle est implémentée ainsi

```
  @IBAction func equal() {
    if !brain.isExpressionCorrect {
      showAlert()
    } else {
      let total = brain.calculateTotal()
      if brain.roundEvaluation(total){
        textView.text! += "\n =(Int(total))"
      } else {
        textView.text! += "\n =(total)"
      }
    }
  }
```

Utilisation du résultat précédent

L'utilisation du résultat précédent se fait via une propriété de type Double
`var formerResult: Double?`

Le stockage

Au moment ou l'utilisateur fait appel à la fonction calculateTotal() la propriété obtient une variable Double de la valeur du résulat.

```
mutating func calculateTotal() -> Double {
    var total: Double = 0
    // slices the memorized number
    for (i, stringNumber) in stringNumbers.enumerated() {
...
    }
    formerResult = total
    clear()
    return total
  }
```

L'utilisation

tout d'abord elle se fait grâce à une computed properties de type Booléen qui verifie si l'on peut ajouter un opérand. Nous faisons donc que si stringNumbers est vide mais formerResult contient un double alors on peut entrer un operand dans la pile des operateur

```
var canAddOperator: Bool {
   if let stringNumber = stringNumbers.last {
     if stringNumber.isEmpty && formerResult == nil{
       return false
     }
   }
   return true
 }
```

L'update de l'écran

Dans le controller,

```
private func updateDisplayForResultReuse(operand: String) {
   updateDisplay()
   brain.sendOperandsToBrain(operand: operand, number: "")
   updateDisplay()
 }
```

Purge des calculs

La purge se fait via le bouton AC -> All Clear. Nous creons donc une fonction qui purge tous les tableaux stringNumbers, operand et la variable former result. Puis update le display pour le reinitialiser.

```
  /**
   Clear the model's data
   */
  mutating func clear() {
    stringNumbers = [String()]
    operators = ["+"]
    index = 0
  }



/**
   Clear the model's data and purge former result
   */
  mutating func allClear() {
    clear()
    formerResult = nil
  }
```

Operation Binaires Supplémentaires

La calculette peut aussi effectuer des opérations binaires supplémentaires tels que multiplier et diviser.

L'implémentaion de l'UI

Nous avons rassembler les boutons sur une ligne dans la vue. Dans le controller, les boutons sont rassemblés dans une outlet collection afin de mutualiser l'action en récupérant uniquement le symbole de l'opération souhaité dit opérateur

Définir une opération

Nous créons une fonctions performOperation() dans le controlleur. Celle-ci permet de faire appel à la logique dans le model et de mettre à jour la vue grace à la fonction update display.

```
/// Perform operation and update display
///
/// - Parameter operand: String representing the mathematical operand
func performOperation(operand:String) {
  if brain.canAddOperator {
    // check if we use the former result
    let result = brain.formerResult
    if result != nil { // if so
      // round result if needed
      brain.roundResult(result)
      // display the operand on textView and send the operand in stack
      updateDisplayForResultReuse(operand: operand)
    } else {
      // send the operand in stack
      brain.sendOperandsToBrain(operand: operand, number: "")
      // display the operand on textView
      updateDisplay()}
  } else {
    showAlert(message: "Expression incorrecte !")
  }
}
```

Le Calcul

Le calcul s'effectue dans le model dans la fonction calculate(). Ce dernier prend les chiffres stockés dans string numbers et effectue une opération sur le nombre affiché.

Nous passons par un switch statement sur le string de l'operateur qui permet de renvoyer vers la opération arithmétique.

```
   switch operators[i]{
      case "+":
        total += number
      case "-":
        total -= number
      case "x":
        total *= number
      case "/":
        total /= number
      default:
        break
      }
```