LINEAS DE PRODUCTOS SOFTWARE

GRUPO GLASS

20 DE ENERO DE 2020
ALEJANDRO FRANCISCO GARCIA UCLES
JUAN SOLER MÁRQUEZ
JOSE MANUAL MARTINEZ SALAS
DIEGO CANGAS MOLDES

UNIVERSIDAD DE ALMERÍA

Proyecto Glass

Linea de Productos Software

Almería, 2 de diciembre de 2019

Sesión: Análisis de datos con Weka

Miembros:
Juan Soler Márquez
Diego Cangas Moldes
Alejandro Francisco García Uclés
José Manuel Martínez Salas

Índice

1. Introducción

Para este análisis hemos utilizado el dataset Glass.arff. Este dataset se trata de un conjunto de datos sobre cristales. Contiene información sobre la composición química de cada uno de los cristales, el índice de refracción, la técnica con la que se ha conformado y la clase a la que pertenece.

2. Modelos caja blanca

2.1. J48

2.1.1. J48 Con todos los atributos

Primero hemos elegido el algoritmo J48 el cual arroja un 66,82 % de instancias clasificadas correctamente.

Para ello cargamos nuestro conjunto de datos *glass.arff* y luego pinchamos en *Classify*, una vez dentro pinchamos en *Choose* y dentro de *tree*, elegimos *J48* pulsamos *Start* y esperamos el resultado. El cual podemos verlo en la *Figura 2*.

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     Glass
Instances:    214
Attributes:   10
              RI
              Na
              Mg
              Al
              Si
              K
              Ca
              Ba
              Fe
              Type
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------
```

Figura 1: Atributos

```
Number of Leaves  :     30

Size of the tree :     59


Time taken to build model: 0.06 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         143               66.8224 %
Incorrectly Classified Instances        71               33.1776 %
Kappa statistic                          0.55
Mean absolute error                      0.1026
Root mean squared error                  0.2897
Relative absolute error                 48.4507 %
Root relative squared error             89.2727 %
Total Number of Instances              214

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,714    0,174    0,667      0,714    0,690      0,532    0,806     0,667     build wind float
               0,618    0,181    0,653      0,618    0,635      0,443    0,768     0,606     build wind non-float
               0,353    0,046    0,400      0,353    0,375      0,325    0,766     0,251     vehic wind float
               ?        0,000    ?          ?        ?          ?        ?         ?         vehic wind non-float
               0,769    0,010    0,833      0,769    0,800      0,788    0,872     0,575     containers
               0,778    0,029    0,538      0,778    0,636      0,629    0,930     0,527     tableware
               0,793    0,022    0,852      0,793    0,821      0,795    0,869     0,738     headlamps
Weighted Avg.  0,668    0,130    0,670      0,668    0,668      0,539    0,807     0,611

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 50 15  3  0  0  1  1 |  a = build wind float
 16 47  6  0  2  3  2 |  b = build wind non-float
  5  5  6  0  0  1  0 |  c = vehic wind float
  0  0  0  0  0  0  0 |  d = vehic wind non-float
  0  2  0  0 10  0  1 |  e = containers
  1  1  0  0  0  7  0 |  f = tableware
  3  2  0  0  0  1 23 |  g = headlamps
```

Figura 2: Resultado

2.1.2. J48 Con selección de atributos

Después de ver los resultados que nos devuelve el J48 con todos los atributos, hemos aplicado una selección de atributos para ver cuáles eran más relevantes y cuáles no. Tras realizar la selección de atributos, nos quedamos con los atributos siguientes:

- Refractive Index
- Magnesium
- Potassium
- Aluminium
- Barium

Con estos atributos, el J48 arroja un 71,028 % de instancias clasificadas correctamente.

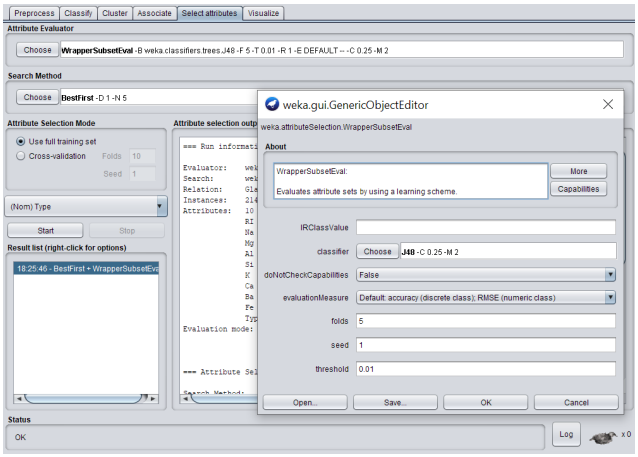


Figura 3: Seleccion de atributos

Para realizar esta operación, entramos en la pestaña *Select Attributes* de *Weka* y en *Attribute Evaluator* seleccionamos *WrapperSubSetEval* y dentro de este en *Classifier* seleccionamos el atributo con el que hemos hecho la clasificación anteriormente para ver qué atributos no son significativos y podemos eliminarlos de nuestro conjunto de datos. Así luego, ejecutaremos de nuevo el algoritmo de clasificación para compararlo con el resultado anterior con todos los atributos.

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     Glass-weka.filters.unsupervised.attribute.Remove-R2,5,7,9
Instances:    214
Attributes:   6
              RI
              Mg
              Al
              K
              Ba
              Type
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

Ba <= 0.27
|   Mg <= 2.41
|   |   K <= 0.03
|   |   |   RI <= 1.52315: tableware (10.0/1.0)
|   |   |   RI > 1.52315: build wind non-float (2.0)
|   |   K > 0.03
|   |   |   Al <= 1.38: build wind non-float (6.0/1.0)
|   |   |   Al > 1.38
|   |   |   |   Mg <= 1.88: containers (14.0/2.0)
|   |   |   |   Mg > 1.88: build wind non-float (3.0/1.0)
|   Mg > 2.41
|   |   Al <= 1.41
|   |   |   RI <= 1.51707
|   |   |   |   RI <= 1.51596: build wind float (3.0)
|   |   |   |   RI > 1.51596
|   |   |   |   |   Al <= 1.27
|   |   |   |   |   |   Mg <= 3.46: vehic wind float (3.0/1.0)
|   |   |   |   |   |   Mg > 3.46: build wind non-float (3.0)
|   |   |   |   |   Al > 1.27: vehic wind float (5.0)
|   |   |   RI > 1.51707
|   |   |   |   K <= 0.23
|   |   |   |   |   Mg <= 3.34: build wind non-float (2.0)
|   |   |   |   |   Mg > 3.34
|   |   |   |   |   |   RI <= 1.52127
|   |   |   |   |   |   |   Al <= 0.91: vehic wind float (5.0/1.0)
|   |   |   |   |   |   |   Al > 0.91: build wind float (5.0/1.0)
|   |   |   |   |   |   RI > 1.52127: build wind float (15.0/1.0)
|   |   |   |   K > 0.23
|   |   |   |   |   Mg <= 3.75
|   |   |   |   |   |   Al <= 1.16
|   |   |   |   |   |   |   RI <= 1.51806: build wind float (7.0/1.0)
|   |   |   |   |   |   |   RI > 1.51806: build wind non-float (5.0)
|   |   |   |   |   |   Al > 1.16: build wind float (37.0/2.0)
|   |   |   |   |   Mg > 3.75: build wind non-float (10.0)
|   |   Al > 1.41: build wind non-float (50.0/10.0)
Ba > 0.27: headlamps (29.0/3.0)
```

Figura 4: Atributos J48sin

```
Number of Leaves  :     19

Size of the tree :      37


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         152               71.028  %
Incorrectly Classified Instances        62               28.972  %
Kappa statistic                          0.6019
Mean absolute error                      0.1019
Root mean squared error                  0.2754
Relative absolute error                 48.0989 %
Root relative squared error             84.8641 %
Total Number of Instances              214

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,829    0,125    0,763      0,829    0,795      0,690    0,860     0,699     build wind float
               0,697    0,181    0,679      0,697    0,688      0,513    0,754     0,600     build wind non-float
               0,235    0,036    0,364      0,235    0,286      0,245    0,766     0,232     vehic wind float
               ?        0,000    ?          ?        ?          ?        ?         ?         vehic wind non-float
               0,692    0,015    0,750      0,692    0,720      0,703    0,821     0,503     containers
               0,556    0,024    0,500      0,556    0,526      0,505    0,873     0,448     tableware
               0,793    0,022    0,852      0,793    0,821      0,795    0,863     0,634     headlamps
Weighted Avg.  0,710    0,113    0,702      0,710    0,704      0,599    0,813     0,596

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 58  9  1  0  0  1  1 |  a = build wind float
  9 53  6  0  3  3  2 |  b = build wind non-float
  6  6  4  0  0  1  0 |  c = vehic wind float
  0  0  0  0  0  0  0 |  d = vehic wind non-float
  0  3  0  0  9  0  1 |  e = containers
  0  4  0  0  0  5  0 |  f = tableware
  3  3  0  0  0  0 23 |  g = headlamps
```

Figura 5: Resultado J48sin

2.2. LMT

2.2.1. LMT con todos los atributos

El algoritmo LMT cuando lo ejecutamos usando todos los atributos arroja un resultado de 68.6916 %.

```
=== Run information ===

Scheme:       weka.classifiers.trees.LMT -I -1 -M 15 -W 0.0
Relation:     Glass
Instances:    214
Attributes:   10
              RI
              Na
              Mg
              Al
              Si
              K
              Ca
              Ba
              Fe
              Type
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Logistic model tree
------------------

Ba <= 0.27
|   Mg <= 2.41: LM_1:31/93 (35)
|   Mg > 2.41
|   |   Al <= 1.41
|   |   |   RI <= 1.51707: LM_2:31/155 (14)
|   |   |   RI > 1.51707
|   |   |   |   K <= 0.23
|   |   |   |   |   Mg <= 3.34: LM_3:0/186 (2)
|   |   |   |   |   Mg > 3.34
|   |   |   |   |   |   Si <= 72.64: LM_4:31/248 (22)
|   |   |   |   |   |   Si > 72.64: LM_5:0/217 (3)
|   |   |   |   K > 0.23
|   |   |   |   |   Mg <= 3.75: LM_6:31/217 (49)
|   |   |   |   |   Mg > 3.75: LM_7:21/207 (10)
|   |   Al > 1.41: LM_8:31/124 (50)
Ba > 0.27: LM_9:31/62 (29)

Number of Leaves  :     9

Size of the Tree :      17
```

Figura 6: Atributos LMT con

```
Time taken to build model: 0.97 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         147               68.6916 %
Incorrectly Classified Instances        67               31.3084 %
Kappa statistic                          0.5702
Mean absolute error                      0.099
Root mean squared error                  0.2737
Relative absolute error                 46.7534 %
Root relative squared error             84.3316 %
Total Number of Instances              214

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,743    0,194    0,650      0,743    0,693      0,532    0,831     0,656     build wind float
               0,671    0,167    0,689      0,671    0,680      0,507    0,802     0,649     build wind non-float
               0,294    0,030    0,455      0,294    0,357      0,323    0,686     0,241     vehic wind float
               ?        0,000    ?          ?        ?          ?        ?         ?         vehic wind non-float
               0,615    0,015    0,727      0,615    0,667      0,650    0,964     0,739     containers
               0,778    0,020    0,636      0,778    0,700      0,689    0,913     0,500     tableware
               0,828    0,016    0,889      0,828    0,857      0,836    0,913     0,845     headlamps
Weighted Avg.  0,687    0,129    0,685      0,687    0,683      0,562    0,832     0,645

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 52 12  4  0  0  1  1 |  a = build wind float
 18 51  2  0  2  2  1 |  b = build wind non-float
  6  5  5  0  0  1  0 |  c = vehic wind float
  0  0  0  0  0  0  0 |  d = vehic wind non-float
  0  4  0  0  8  0  1 |  e = containers
  1  1  0  0  0  7  0 |  f = tableware
  3  1  0  0  1  0 24 |  g = headlamps
```

Figura 7: Resultado LMT con

2.2.2. LMT con selección de atributos

Después de ver los resultados que nos devuelve el LMT con todos los atributos, hemos aplicado una selección de atributos para ver cuáles eran más relevantes y cuáles no. Tras realizar la selección de atributos, nos quedamos con los atributos siguientes:

- Refractive Index
- Magnesium
- Potassium
- Aluminium
- Barium
- Calcium

Con estos atributos, el LMT arroja un 66.8224 % de instancias clasificadas correctamente.

```
=== Run information ===

Scheme:       weka.classifiers.trees.LMT -I -1 -M 15 -W 0.0
Relation:     Glass-weka.filters.unsupervised.attribute.Remove-R2,5,9
Instances:    214
Attributes:   7
              RI
              Mg
              Al
              K
              Ca
              Ba
              Type
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Logistic model tree
------------------

Ba <= 0.27
|   Mg <= 2.41: LM_1:12/36 (35)
|   Mg > 2.41
|   |   Al <= 1.41
|   |   |   RI <= 1.51707: LM_2:12/60 (14)
|   |   |   RI > 1.51707
|   |   |   |   K <= 0.23: LM_3:12/72 (27)
|   |   |   |   K > 0.23: LM_4:12/72 (59)
|   |   Al > 1.41
|   |   |   Mg <= 3.45: LM_5:12/60 (15)
|   |   |   Mg > 3.45: LM_6:12/60 (35)
Ba > 0.27: LM_7:12/24 (29)

Number of Leaves  :     7

Size of the Tree :      13
```

Figura 8: Atributos LMTsin

```
Time taken to build model: 0.17 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         143               66.8224 %
Incorrectly Classified Instances        71               33.1776 %
Kappa statistic                          0.5465
Mean absolute error                      0.1027
Root mean squared error                  0.2892
Relative absolute error                 48.5161 %
Root relative squared error             89.1214 %
Total Number of Instances              214

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,743    0,188    0,658      0,743    0,698      0,540    0,810     0,590     build wind float
               0,632    0,174    0,667      0,632    0,649      0,464    0,753     0,607     build wind non-float
               0,294    0,051    0,333      0,294    0,313      0,258    0,716     0,236     vehic wind float
               ?        0,000    ?          ?        ?          ?        ?         ?         vehic wind non-float
               0,615    0,010    0,800      0,615    0,696      0,685    0,953     0,602     containers
               0,778    0,015    0,700      0,778    0,737      0,726    0,933     0,495     tableware
               0,793    0,027    0,821      0,793    0,807      0,778    0,917     0,758     headlamps
Weighted Avg.  0,668    0,132    0,668      0,668    0,666      0,539    0,811     0,587

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 52 12  4  0  0  1  1 |  a = build wind float
 16 48  6  0  2  1  3 |  b = build wind non-float
  6  5  5  0  0  1  0 |  c = vehic wind float
  0  0  0  0  0  0  0 |  d = vehic wind non-float
  0  4  0  0  8  0  1 |  e = containers
  1  1  0  0  0  7  0 |  f = tableware
  4  2  0  0  0  0 23 |  g = headlamps
```

Figura 9: Resultado LMTsin

2.3. Conclusiones

Una vez analizados los resultados con los algoritmos J48 y LMT, hemos decidido usar el algoritmo J48 para implementar el algoritmo de caja blanca en nuestra aplicación. Las razones por la cual hemos elegido ésta es porque el porcentaje de instancias clasificadas correctamente es mejor y el árbol de decisión resultante es bastante más simple que el árbol que genera el algoritmo LMT.

3. Modelo caja negra

3.1. MLP

3.1.1. MLP con todos los atributos

El algoritmo MLP, de caja negra, cuando lo ejecutamos usando todos los atributos arroja un resultado de clasificación correcta de un 67.757 %.

```
=== Run information ===

Scheme:       weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Relation:     Glass
Instances:    214
Attributes:   10
              RI
              Na
              Mg
              Al
              Si
              K
              Ca
              Ba
              Fe
              Type
Test mode:    10-fold cross-validation
```

Figura 10: Atributos MLP con

```
Time taken to build model: 0.57 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         145               67.757  %
Incorrectly Classified Instances        69               32.243  %
Kappa statistic                          0.5528
Mean absolute error                      0.1114
Root mean squared error                  0.2627
Relative absolute error                 52.5915 %
Root relative squared error             80.958  %
Total Number of Instances              214

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,743    0,181    0,667      0,743    0,703      0,548    0,862     0,643     build wind float
                 0,684    0,217    0,634      0,684    0,658      0,460    0,815     0,741     build wind non-float
                 0,059    0,005    0,500      0,059    0,105      0,151    0,654     0,161     vehic wind float
                 ?        0,000    ?          ?        ?          ?        ?         ?         vehic wind non-float
                 0,692    0,030    0,600      0,692    0,643      0,620    0,957     0,604     containers
                 0,778    0,020    0,636      0,778    0,700      0,689    0,926     0,514     tableware
                 0,828    0,011    0,923      0,828    0,873      0,856    0,931     0,889     headlamps
Weighted Avg.    0,678    0,141    0,671      0,678    0,659      0,537    0,847     0,665

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 52 16  1  0  0  0  1 |  a = build wind float
 17 52  0  0  4  3  0 |  b = build wind non-float
  8  8  1  0  0  0  0 |  c = vehic wind float
  0  0  0  0  0  0  0 |  d = vehic wind non-float
  0  3  0  0  9  0  1 |  e = containers
  0  1  0  0  1  7  0 |  f = tableware
  1  2  0  0  1  1 24 |  g = headlamps
```

Figura 11: Resultado MLP con

3.1.2. MLP con selección de atributos

Como podemos ver el resultado con la selección de atributos empeora, por lo tanto para nuestro modelo de caja negra, utilizaremos todos los atributos del conjunto de datos.

```
=== Run information ===

Scheme:       weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Relation:     Glass-weka.filters.unsupervised.attribute.Remove-R5,7
Instances:    214
Attributes:   8
              RI
              Na
              Mg
              Al
              K
              Ba
              Fe
              Type
Test mode:    10-fold cross-validation
```

Figura 12: Atributos MLP sin

```
Time taken to build model: 0.35 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         141               65.8879 %
Incorrectly Classified Instances        73               34.1121 %
Kappa statistic                          0.5238
Mean absolute error                      0.1179
Root mean squared error                  0.2721
Relative absolute error                 55.6679 %
Root relative squared error             83.828  %
Total Number of Instances              214

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,671    0,181    0,644      0,671    0,657      0,486    0,835     0,581     build wind float
                 0,724    0,254    0,611      0,724    0,663      0,456    0,808     0,699     build wind non-float
                 0,000    0,005    0,000      0,000    0,000      -0,020   0,607     0,123     vehic wind float
                 ?        0,000    ?          ?        ?          ?        ?         ?         vehic wind non-float
                 0,692    0,025    0,643      0,692    0,667      0,645    0,952     0,722     containers
                 0,778    0,015    0,700      0,778    0,737      0,726    0,958     0,716     tableware
                 0,793    0,016    0,885      0,793    0,836      0,814    0,949     0,817     headlamps
Weighted Avg.    0,659    0,154    0,616      0,659    0,635      0,499    0,835     0,633

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 47 21  1  0  0  0  1 |  a = build wind float
 16 55  0  0  2  2  1 |  b = build wind non-float
  9  8  0  0  0  0  0 |  c = vehic wind float
  0  0  0  0  0  0  0 |  d = vehic wind non-float
  0  3  0  0  9  0  1 |  e = containers
  0  1  0  0  1  7  0 |  f = tableware
  1  2  0  0  2  1 23 |  g = headlamps
```

Figura 13: Resultado MLP sin

3.2. Conclusiones

Para nuestro modelo de caja negra, hemos decidido usar el algoritmo MLP con todos los atributos, ya que al hacer uso de la selección de atributos el porcentaje de instancias clasificadas correctamente disminuye. Por ello, en la interfaz de nuestra aplicación, cuando tengamos que introducir datos para algoritmo caja negra tendremos disponibles todos los atributos, en cambio, cuando introduzcamos valores para el algoritmo de caja blanca, solo aparecerán activos los atributos seleccionados en el paso 2.1.2


ABC
09:22 PM
Version 1.0
Email
Password
Registe
Sign In


ABC
09:23 PM
Logout
Cases
+
Case: El caso del gallinero
SampleNr: 15 Created: 17.1.2020

ABC 09:31 PM

Casses Members Save

joseJua@hotmail.com

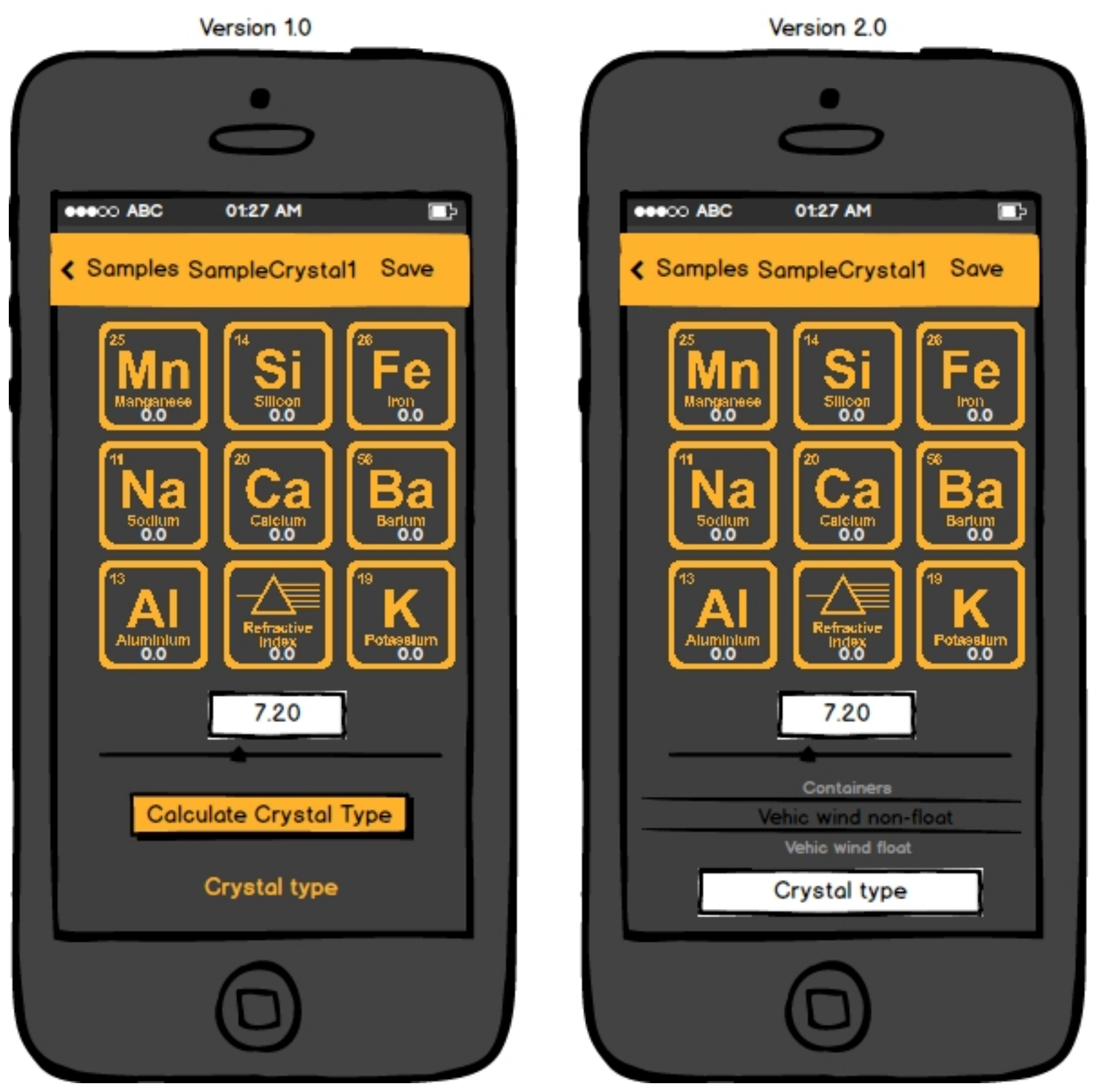
FranccoFr@hotmail.com

Elyoni@hotmail.com

Administrador


ABC
12:26 AM
Cases
Samples
+
Team Update
SampleCrystal1
CrystalType: Tableware
SampleCrystal2
CrystalType: Vehic wind non-floc

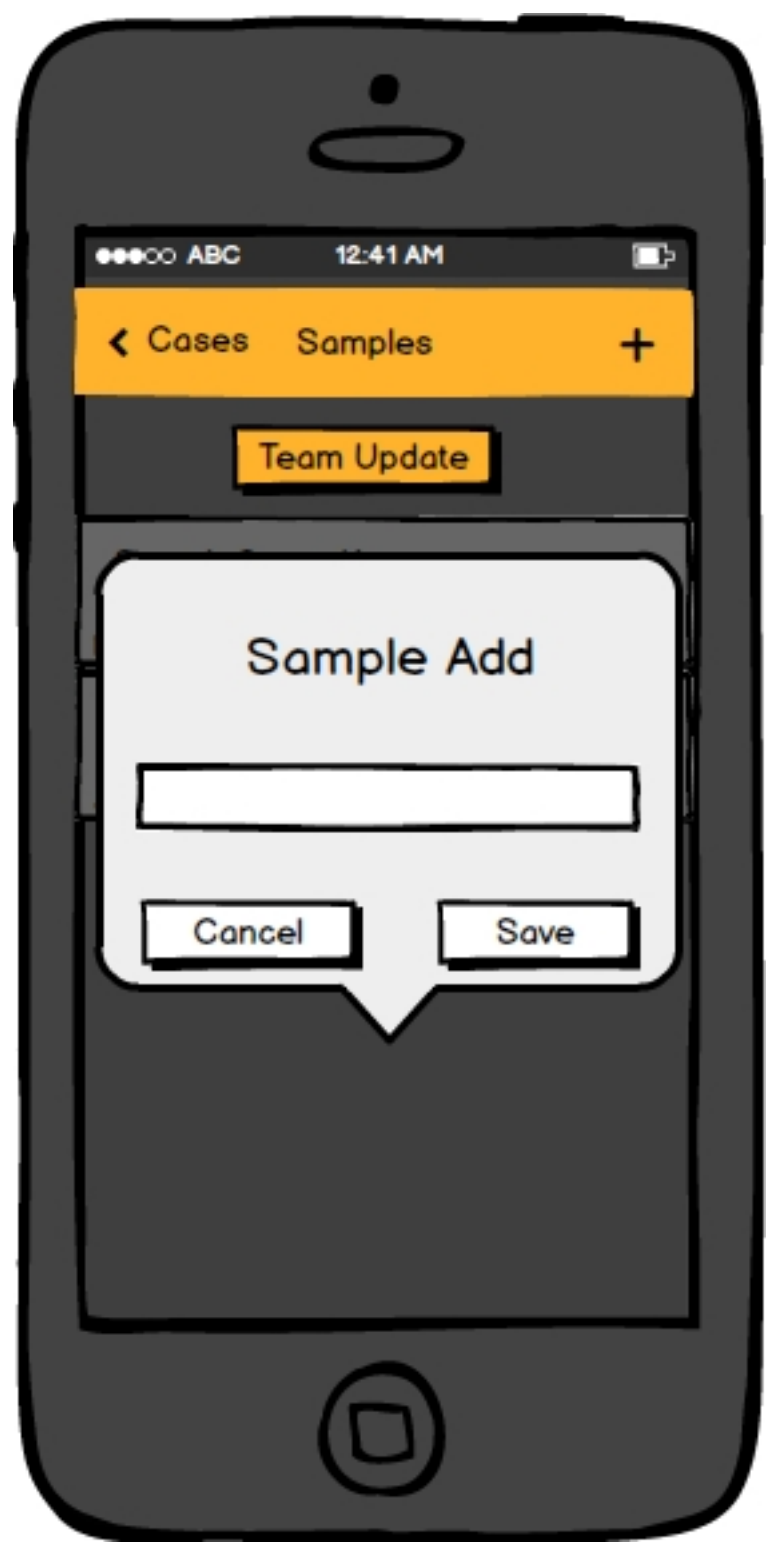

ABC
09:33 PM
Cases
New case
Save
Case Name
Add members


Version 1.0
Version 2.0
ABC
01:27 AM
Samples SampleCrystal1 Save
25 Mn Manganese 0.0
14 Si Silicon 0.0
26 Fe Iron 0.0
11 Na Sodium 0.0
20 Ca Calcium 0.0
56 Ba Barium 0.0
13 Al Aluminium 0.0
Refractive Index 0.0
19 K Potassium 0.0
7.20
Calculate Crystal Type
Crystal type
Containers
Vehic wind non-float
Vehic wind float

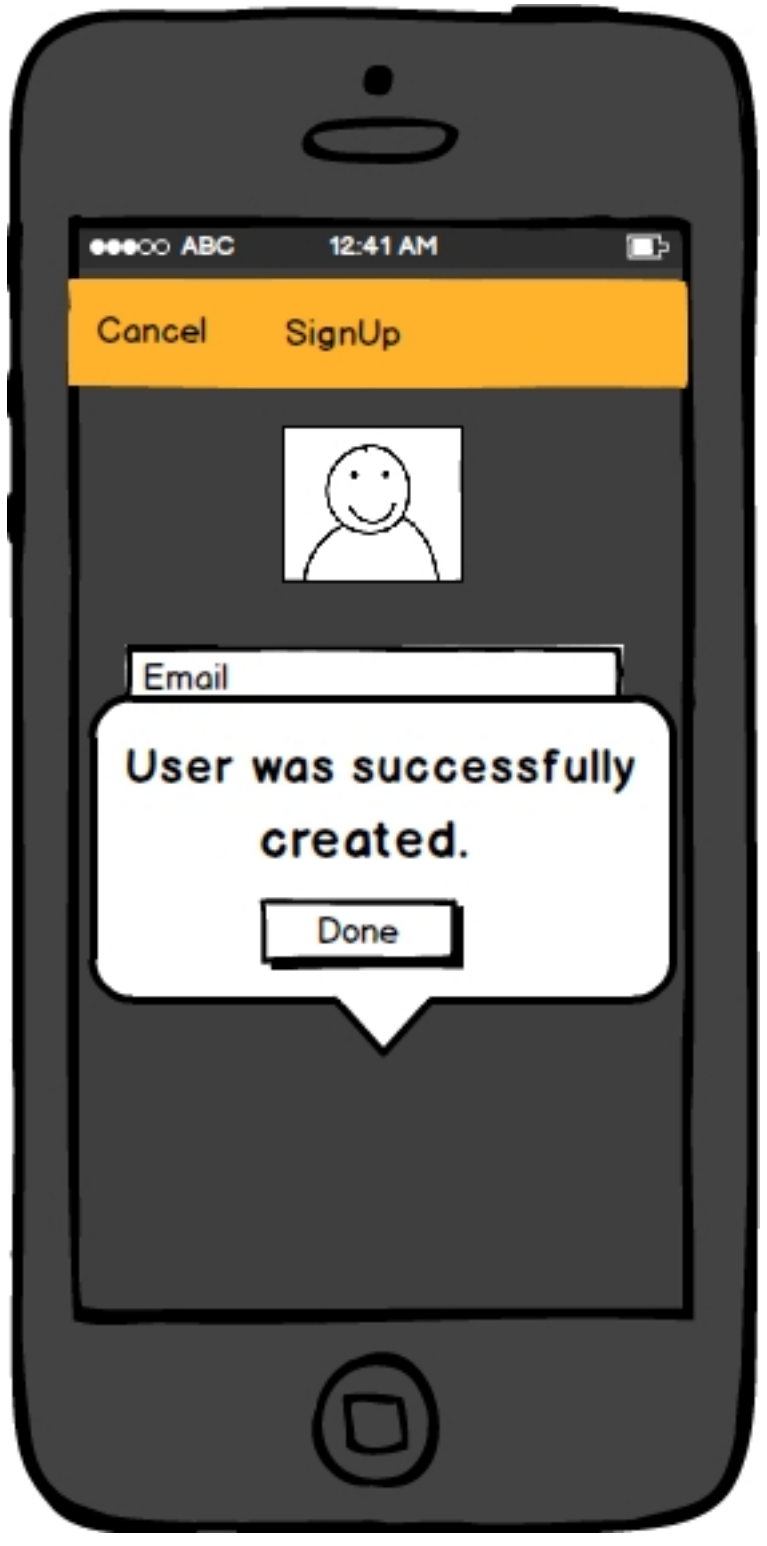

ABC
09:44 PM
Cases
Add member
Save
Email
Admin

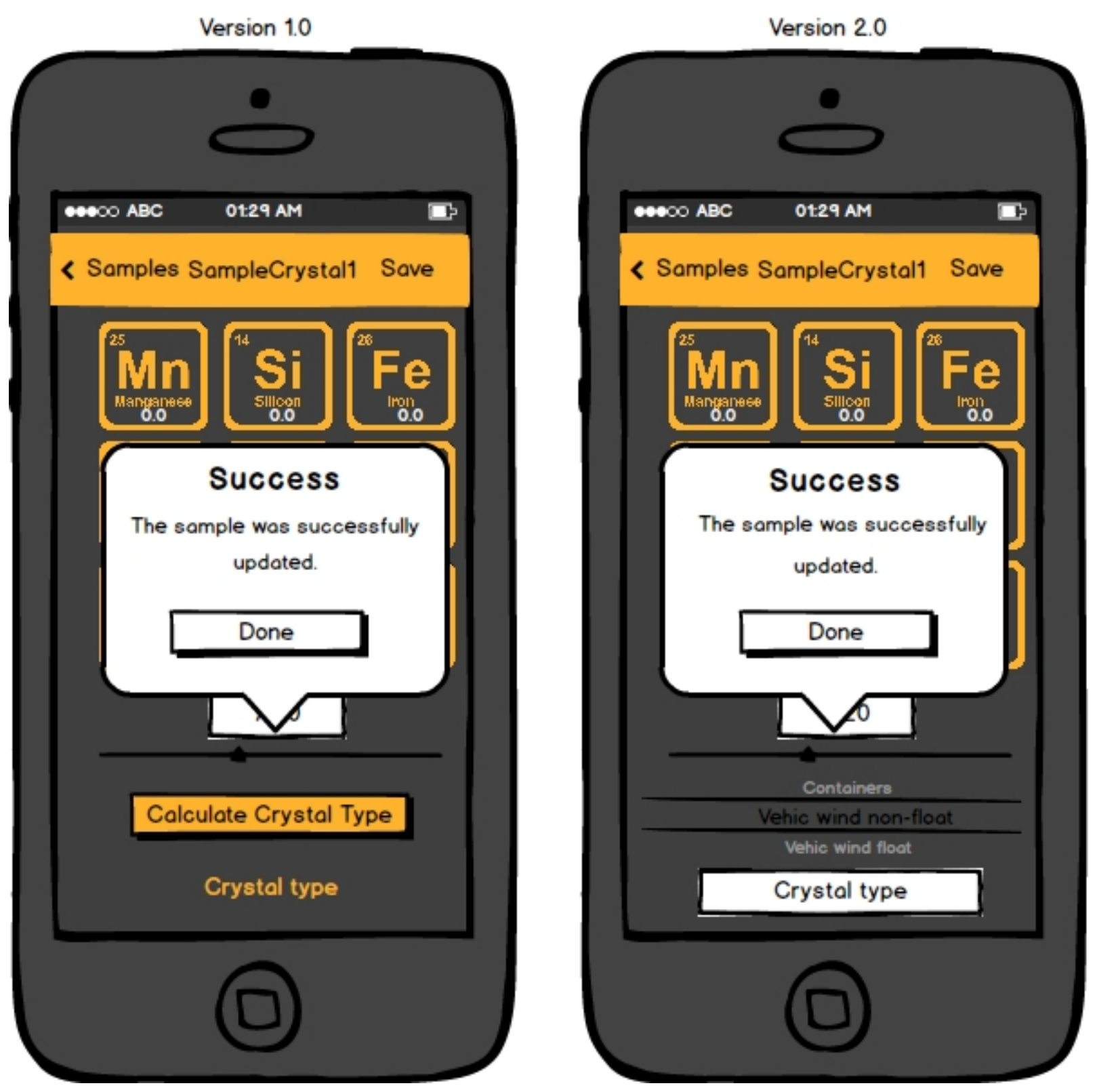

ABC
09:31 PM
Logout
Admin
User Mode
Add Case
Add User


ABC
09:49 PM
Cancel
SignUp
Email
Password
ImageUp
SignUp


ABC
12:41 AM
Cases
Samples
+
Team Update
Sample Add
Cancel
Save


ABC
12:41 AM
Cases
Add member
Save
Email
Admin
Success
Member was successfully created.
Done


ABC
12:41 AM
Cancel
SignUp
Email
User was successfully created.
Done


Version 1.0
ABC
01:29 AM
Samples
SampleCrystal1
Save
25
Mn
Manganese
0.0
14
Si
Silicon
0.0
26
Fe
Iron
0.0
Success
The sample was successfully updated.
Done
Calculate Crystal Type
Crystal type
Version 2.0
ABC
01:29 AM
Samples
SampleCrystal1
Save
25
Mn
Manganese
0.0
14
Si
Silicon
0.0
26
Fe
Iron
0.0
Success
The sample was successfully updated.
Done
Containers
Vehic wind non-float
Vehic wind float
Crystal type

Aspectos destacados de implementación

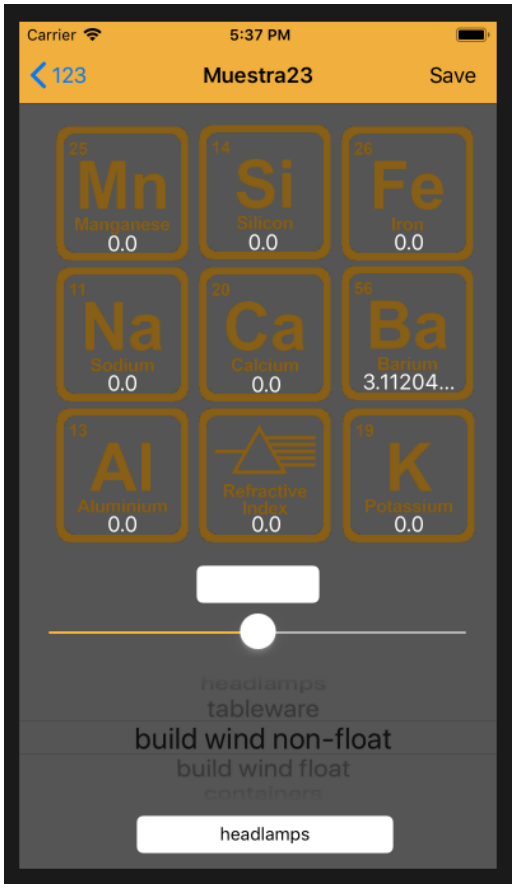
En este apartado se comentarán algunos aspectos destacados sobre la implementación de nuestra aplicación.

Esta aplicación tiene como objetivo facilitar la información sobre muestras en investigaciones sobre tipos de cristales a través de su composición química.

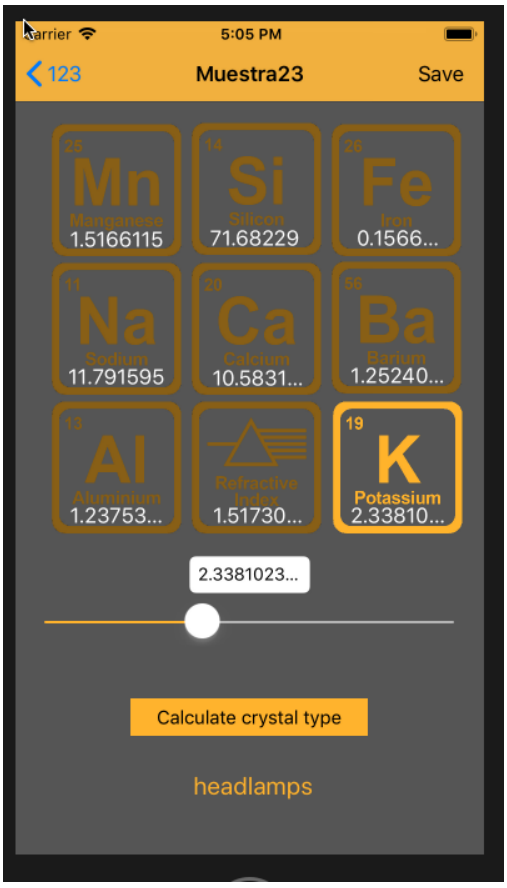
Esta aplicación está dividida en dos versiones 1.0 y 2.0. Las diferencias entre la versión 1.0 y 2.0 son que en la versión 1.0, el investigador elije el tipo de cristal al que pertenece la muestra mientras que en la versión 2.0 el tipo de cristal que es se calcula mediante su composición química.

Esto se lleva a cabo en la interfaz de detalles de muestra donde introduciendo los componentes esta misma te calcula el tipo de cristal que es.

Versión 1.0



Versión 2.0

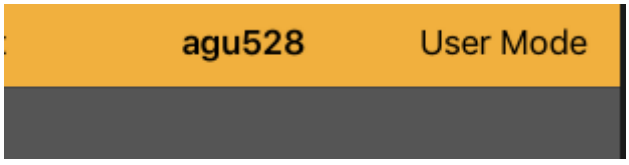


En esta ventana de la interfaz, cada elemento químico y el índice de refracción son un botón que, al pulsarlo, cambia el límite del slider(campo de desplazamiento) justo debajo del textfield (campo de texto) con el valor de ejemplo “7.20”. Esto se debe a que cada elemento químico para nuestro caso tendrá uno mínimo y un máximo distinto. El campo de texto que hemos pasado por encima antes, muestra el valor del elemento químico señalado ahora mismo, pudiéndose cambiar el valor de este campo de texto y afectando al mismo químico.

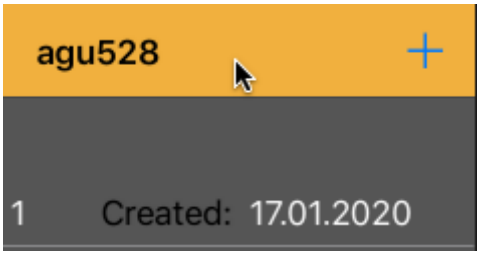
La barra deslizante tiene como objetivo facilitar la introducción de valores. Como dijimos antes, en la versión 1 elegiremos el tipo de cristal con nuestro picker, pero en la versión dos pulsando el botón “Calculate Crystal Type” nos devuelve el tipo de cristal justo en el campo de texto no editable de abajo.

Otro aspecto destacado en nuestra implementación es la posibilidad de que el administrador acceda al modo usuario. Desde este modo, el administrador puede llevar a cabo la adición de nuevos casos mientras que todavía puede ver lo mismo que ve un usuario.

Entrar a modo usuario



Añadir nuevo caso desde modo usuario



En nuestra aplicación, tenemos tres tipos de usuarios:

1. Usuario admin
2. Usuario miembro
3. Usuario no miembro

Estos tipos de usuario han sido listados jerárquicamente, de más poder en la aplicación a menos. La razón de que se añada en aspectos destacados reside en la posibilidad de no ser miembro.

¿Qué es un miembro? Un miembro es un usuario que ha sido validado por un usuario administrador por medio del “Add User”.

¿Por qué esta explicación? Todo el mundo puede registrarse en nuestra aplicación, pero solo aquellos que estén validados como miembros por un usuario administrador podrán ser parte de los casos. Y ya que para ver un caso, un usuario administrador debe haberte incluido en la lista de miembros de ese caso, nunca podrías ver caso alguno ya que ni siquiera podrías ser incluido en estos.

Nuestra aplicación cuenta también con un sistema que mantiene al usuario logueado aunque abandone la aplicación. Esto viene de parte de Firebase, un sistema de bases de datos en línea que hemos decidido usar para nuestra aplicación.

Y puesto que Firebase será nuestra base de datos, lo conveniente es dar una buena explicación.

Lo primero que necesita un proyecto Swift para que se conecte con Firebase es agregar los SDK de Firebase a nuestra App y eso se hace usando la herramienta CocoaPods, es parecido a npm. En nuestro archivo PodFile, agregamos los paquetes que queremos agregar a nuestro proyecto. Por ejemplo, si queremos usar el servicio de autentificación de Firebase debemos agregar la siguiente línea:

```
pod 'Firebase/Auth'
```

De este modo ya podremos hacer import FirebaseAuth y poder hacer uso de los métodos Auth.auth().signIn o Auth.auth().createUser. y poder loguearnos, crear usuarios, etc.

Es muy importante inicializar nuestra conexión con Firebase antes de hacer uso de cualquiera de sus servicios, para ello debemos agregar FirebaseApp.configure() en nuestro AppDelegate en func application, para que lo inicialice al arrancar nuestra aplicación.

Otro paso fundamental es agregar el archivo GoogleService-Info.plist el cual lo podemos descargar directamente desde el apartado de configuración de nuestro proyecto en Firebase. Este archivo contiene identificadores de proyecto de Firebase para enlazar nuestra aplicación a nuestro proyecto en Firebase.

Para hacer uso del servicio de base de datos de Firebase, tenemos que agregar el pod ‘Firebase/Database’, gracias a este import podremos llamar a Database.database().reference(withPath: "nombre_de_la_coleccion") y con ello ya podremos conseguir los hijos de la colección que queremos o bien crear hijos nuevos, actualizarlos o eliminarlos.

Con Firebase podemos modelar objetos para que sea mas fácil hacer CRUD en la base de datos, olvidándonos de crear sentencias personalizadas para actualizar algún atributo de nuestro objeto.

Para poder hacer esto, creamos un modelo e importamos Firebase y FirebaseDatabase, ahora ya podemos crear un constructor que pase una referencia de un objeto de ese tipo que este alojado en Firebase y lo convertirá automáticamente a un objeto compatible para que podamos usarlo y llamar a todos los atributos o métodos que hayamos definido en él. Obviamente, también tendrá su constructor clásico para poder inicializar un objeto pasándole los parámetros que hayamos definido en él (nombre, fecha, addedByUser, etc).

```
init(name: String, addedByUser: String, completed: Bool, key: String =
    "", fecha: String) {
    self.ref = nil
    self.key = key
    self.name = name
    self.addedByUser = addedByUser
    self.completed = completed
    self.fecha = fecha
}

init?(snapshot: DataSnapshot) {
    guard
        let value = snapshot.value as? [String: AnyObject],
        let name = value["name"] as? String,
        let addedByUser = value["addedByUser"] as? String,
        let fecha = value["fecha"] as? String,
        let completed = value["completed"] as? Bool else {
            return nil
    }

    self.ref = snapshot.ref
    self.key = snapshot.key
    self.name = name
    self.addedByUser = addedByUser
    self.completed = completed
    self.fecha = fecha
}
```

Para hacer la función inversa, convertir nuestro objeto en un objeto tipo JSON que es lo que necesita Firebase para agregar la información en la base de datos creamos este método en nuestro modelo.

```
func toAnyObject() -> Any {
    return [
        "name": name,
        "addedByUser": addedByUser,
        "completed": completed,
        "fecha": fecha
    ]
}
```

En la siguiente imagen, podemos ver como creamos un objeto de tipo Case llamado caseItem. Luego creamos un hijo y guardamos su referencia en caseItemRef y finalmente agregamos el objeto Case a ese hijo con setValue y le pasamos el toAnyObject para que tenga la forma JSON deseada por Firebase.

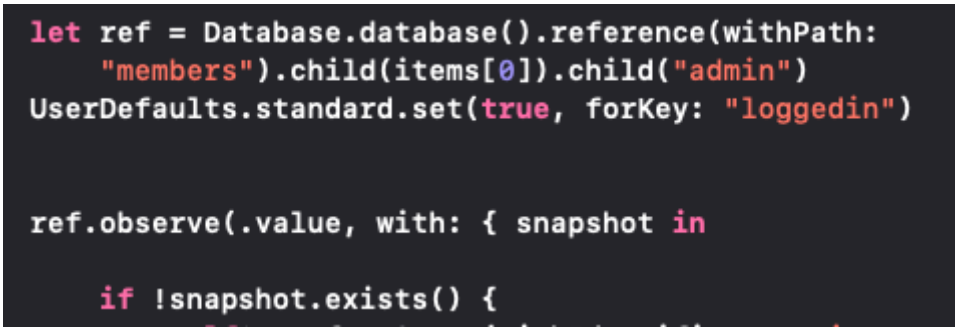
```
let caseItem = Case(name: nameText.text!,
                    addedByUser: AppDelegate.user!.email,
                    completed: false, fecha:formatter.string(from:
                        date))

let caseItemRef = Database.database().reference(withPath: "case-
    items").child(nameText.text!.lowercased())


caseItemRef.setValue(caseItem.toAnyObject())
```

Si queremos ver si un hijo existe, podemos hacer uso del método observe, el cual recibe los valores que tenga ese hijo y podemos comprobar si existe, recorrerlo, etc.

En la siguiente imagen, comprobamos si existe y si existe ya podemos obtener su valor con snapshot.value o incluso si tiene hijo a su vez, con snapshot.children.



En la siguiente imagen vemos como recorremos los hijos de un hijo, en este caso, dentro de la colección “members” hay un hijo llamado “cases”. Como podemos ver, podemos pedirle a Firebase que nos devuelva a los hijos ordenados por algún atributo y puede ser ASC o DESC. Una vez que empezamos a recorrer a los hijos y obtenemos objetos de tipo DataSnapshot, recordamos el constructor que creamos en nuestro modelo el cual recibia como parámetro un DataSnapShot y que el cual pudimos establecerlo gracias al import FirebaseDatabase que pusimos en nuestra clase.

Al llamar a este constructor, automáticamente convierte el Snapshot en un objeto de tipo Case perfectamente valido para poder trabajar con él.

```
let ref = Database.database().reference(withPath: "members")

    ref.child(AppDelegate.username!).child("cases").queryOrdered
    (byChild: "completed").observe(.value, with: { snapshot in
    var newItems: [Case] = []
    for child in snapshot.children {
        if let snapshot = child as? DataSnapshot,
            let caseItem = Case(snapshot: snapshot) {
            print(caseItem.name)
            newItems.append(caseItem)
        }
    }

    self.items = newItems
    self.tableView.reloadData()
})
```

Una de las funcionalidades a destacar de Firebase es el uso de Observables, gracias a ello, cualquier cambio que se realice en nuestra base de datos, se actualizará automáticamente en nuestra aplicación y todo ello “gratis” sin tener que llamar a listeners, valueChanged u otros métodos que estén atentos para ver si ha habido algún cambio en nuestra base de datos.