



UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba



Sistemas Interactivos

Aplicación Manage your Home

Juan José Méndez Torrero
i42metoj@uco.es

Universidad de Córdoba

12 de junio de 2019

Índice

| | |
|--|-----------|
| 1. Introducción e identificación del problema | 3 |
| 2. Especificación de requisitos | 6 |
| 2.1. Requisitos de usuario | 6 |
| 2.2. Requisitos de sistema | 6 |
| 3. Diseño y especificación de la interfaz | 7 |
| 3.1. Página de inicio | 7 |
| 3.2. Pantalla principal | 9 |
| 3.3. Pantalla de dispositivos | 10 |
| 3.4. Pantalla de nuevo dispositivo | 11 |
| 3.5. Pantalla de edición | 12 |
| 3.6. Pantalla Mapa | 13 |
| 3.7. Pantalla de ayuda | 14 |
| 4. Implementación | 16 |
| 4.1. Clase Lamp | 16 |
| 4.2. Clase Lock | 16 |
| 4.3. Clase Air | 16 |
| 4.4. Clase Blind | 16 |
| 4.5. Clase Mail | 17 |
| 4.6. Clases de nuevo dispositivo | 17 |
| 4.7. Clases de edición de dispositivo | 18 |
| 4.8. Clase Devices | 20 |
| 4.9. Clase Map | 23 |
| 5. Futuras mejoras | 26 |

1. Introducción e identificación del problema

Hoy en día, la presencia de dispositivos electrónicos inteligentes en los hogares está en aumento. Esto hace que los usuarios no expertos tengan que aprender cómo conectar sus dispositivos a la red para poder hacer uso de ellos. Además, estos dispositivos, al ser cada uno de una empresa distinta, tienen cada uno una interfaz distinta, haciendo que los usuarios tengan que cambiar de aplicación cada vez que quieran hacer uso de algún dispositivo.

Cuando el usuario no experto tiene que realizar cambios en la red o cambios en las interfaces de cada uno de sus dispositivos, hace que la probabilidad de cometer un fallo aumente, con lo que ese usuario no experto necesitará la ayuda de un técnico experto para la configuración de estos dispositivos.

Estos dispositivos a lo largo de los años han aumentado, siendo las cafeteras consideradas como dispositivos inteligentes, ya que existen cafeteras que, programándolo a través de una interfaz, nos pueden preparar un café según la hora o el día que hayamos programado. Como ejemplo, tenemos una cafetera de *BOSCH* [1] la cual puede ser programada para preparar un café una vez nos hayamos despertado.

Para resolver los problemas de usabilidad de los usuarios no expertos, hemos creado esta aplicación la cuál permitirá a estos usuarios controlar, de manera sencilla y rápida los siguientes dispositivos:

- Lámparas inteligentes. En esta aplicación el usuario podrá conectar dos lámparas inteligentes y controlarlas desde una sola aplicación.
- Cerrojos inteligentes. El usuario podrá conectar dos cerrojos a la aplicación y controlarlos haciendo click en ellos.
- Aire acondicionado inteligente. El usuario podrá conectar y usar dos aires acondicionados inteligentes desde esta aplicación.
- Persianas inteligentes. El usuario puede abrir o cerrar dos persianas inteligentes que estén conectadas a la red de su casa.

Como antecedentes a esta aplicación, podemos encontrarnos con las siguientes aplicaciones:

- **Philips Hue:** Esta aplicación [2] permite controlar los dispositivos *Philips* inteligentes de tu casa. En esta aplicación se podrá realizar el control de los dispositivos de manera rápida y sencilla, pudiendo clasificar los dispositivos por habitación o por tipo. Por último, esta aplicación incluye un control por voz de los dispositivos a través de ésta.
- **Home Intelligence:** Esta aplicación fue creada por alumnos de la Universidad de Córdoba en ella podemos controlar los dispositivos inteligentes de nuestra casa de manera visual.
- **Google Home:** Esta aplicación, creada por Google, nos permite configurar y controlar los dispositivos inteligentes de nuestro hogar de manera muy sencilla, ya que el usuario no experto no necesitará saber sobre conexiones red para poder añadir un nuevo dispositivo inteligente a su hogar, ya que el reconocimiento de estos dispositivos se hace de manera automática, detectándolos de tal manera que el usuario sólo tenga que señalar cuál es el dispositivo que quiere añadir.
- **Hestia iOS:** Aplicación [3] creada por alumnos de la universidad de Groningen que se encarga de dar la posibilidad al usuario de poder tener en la misma aplicación una lámpara de una empresa junto con otra lámpara de otra empresa distinta. Además, esta aplicación permite el reconocimiento por voz de comandos para controlar todos los dispositivos.

Finalmente, hemos creado esta aplicación para satisfacer los siguientes objetivos:

- Un usuario no experto no necesitará conocimientos sobre conexiones red para poder añadir un nuevo dispositivo inteligente a su hogar.
- El usuario no necesitará tener una interfaz distinta según la compañía de cada dispositivo, si no que, con una misma aplicación podrá controlar dispositivos que son de diferentes compañías.

- Permitir al usuario controlar sus dispositivos con sólo uno o dos pasos.
- El usuario podrá ver en la misma pantalla todos los dispositivos conectados y cuáles están en uso.

2. Especificación de requisitos

En esta sección se abordará la cuestión de cuáles son los requisitos de usuario y cuáles son los requisitos de sistema.

2.1. Requisitos de usuario

- **RU-1:** La aplicación debe permitir configurar un nuevo dispositivo considerando sólo la dirección IP del dispositivo.
- **RU-2:** El usuario podrá eliminar los dispositivos inteligentes configurados de manera sencilla.
- **RU-3:** El usuario podrá conectarse a la aplicación ya sea con su nombre de usuario, su correo electrónico o cuenta de Google.
- **RU-4:** Se permite al usuario editar cada uno de los dispositivos.
- **RU-5:** El usuario conocerá en tiempo real cuáles son los dispositivos conectados y usados.

2.2. Requisitos de sistema

- **RS-1:** La aplicación debe permitir añadir un nuevo dispositivo si la cantidad máxima de dispositivos por categoría no ha sido alcanzada.
- **RS-2:** La aplicación debe permitir eliminar un dispositivo si éste dispositivo está conectado a la aplicación.
- **RS-3:** La aplicación permitirá al usuario crear una nueva cuenta para poder hacer uso de ésta.
- **RS-4:** La aplicación estará disponible para PC, siendo indiferente el sistema operativo usado.

3. Diseño y especificación de la interfaz

En esta sección se explicará el diseño de cada una de las secciones de esta aplicación. Además, se incluirán imágenes ilustrativas para poder explicar cómo funciona la aplicación.

3.1. Página de inicio

En esta pantalla el usuario podrá entrar a la aplicación ya sea con su correo, nombre de usuario o a través de su cuenta de Google. En la Figura 1 se puede observar dicha pantalla.

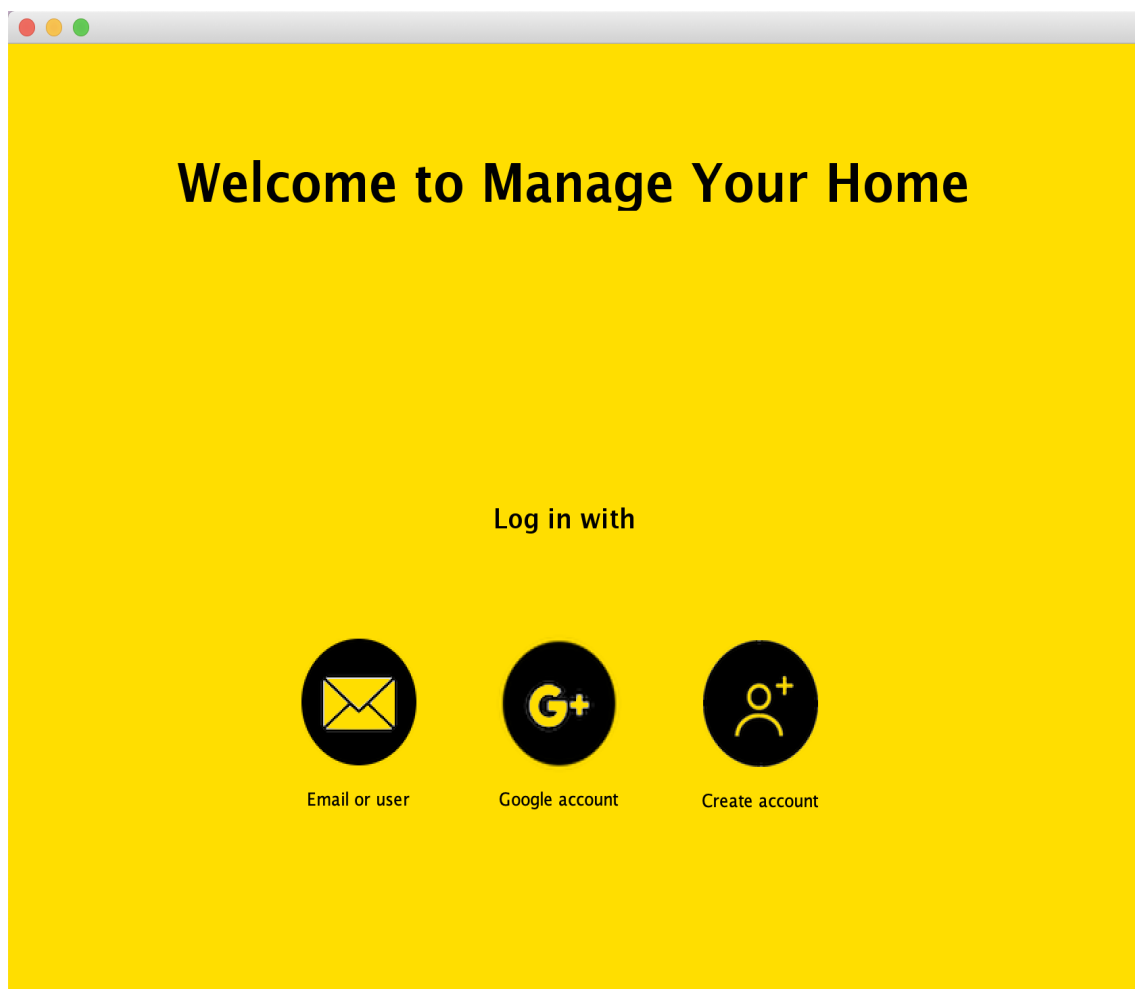
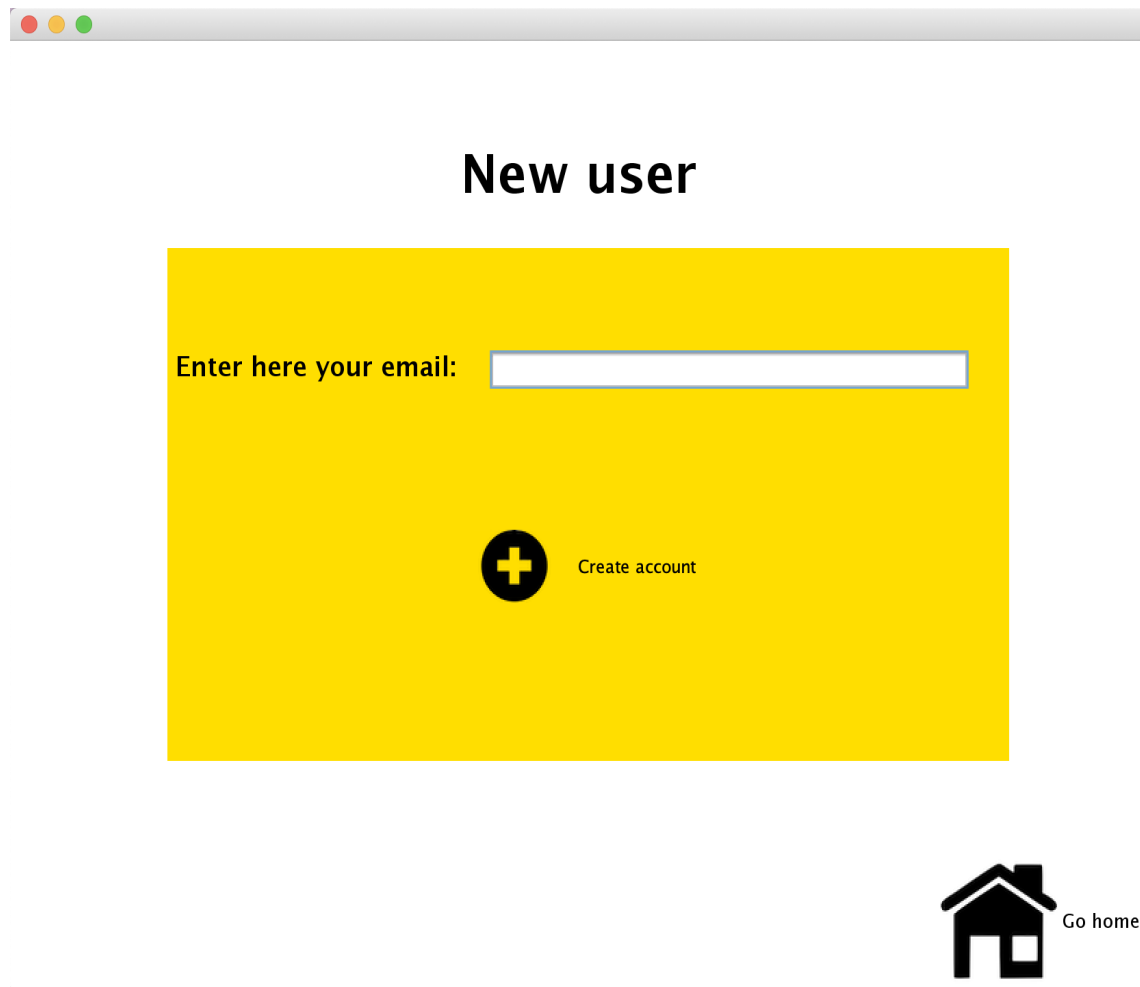


Figura 1: Página de inicio


Como podemos observar, si el usuario no tiene la posibilidad de entrar con ninguna de las cuentas anteriores, el usuario podrá crearse una nueva cuenta para poder entrar. En la Figura 2 se muestra la pantalla en la

que el usuario podrá crear su nueva cuenta.



New user

Enter here your email:

 Create account


 Go home

Figura 2: Pantalla de nuevo usuario

Una vez creada la cuenta, el usuario ya podrá introducir su nombre o correo electrónico, dependiendo de cómo se haya inscrito, y entrar a la aplicación para poder controlar sus dispositivos inteligentes. En la Figura 3 se muestra la pantalla correspondiente a la selección de nombre de usuario o correo electrónico.

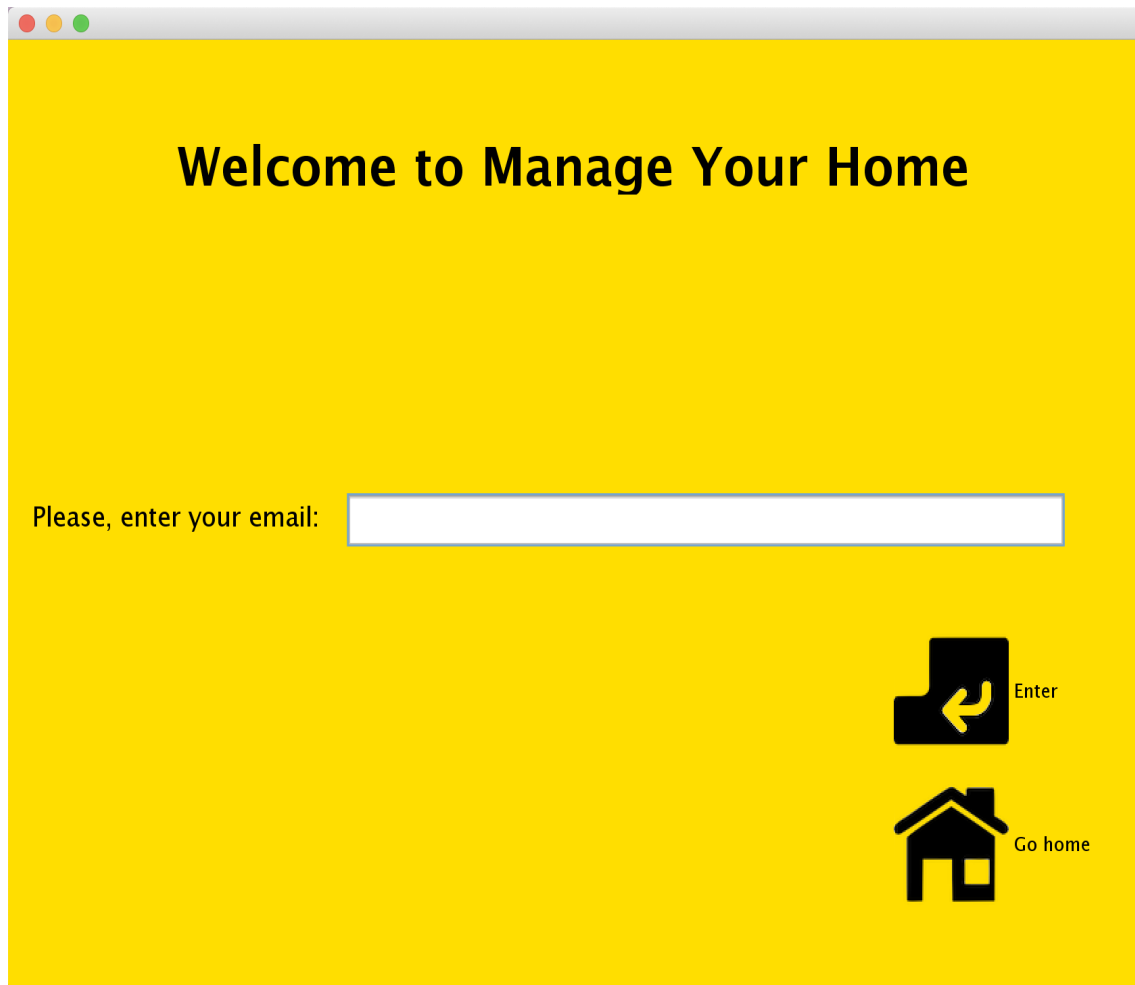


Figura 3: Página de selección de usuario

3.2. Pantalla principal

Una vez el usuario ha introducido su cuenta o nombre de usuario, éste llegará a la pantalla principal. En esta pantalla se da la bienvenida al usuario y se explica brevemente en qué consiste la aplicación. En la Figura 4 podemos observar la pantalla resultante.

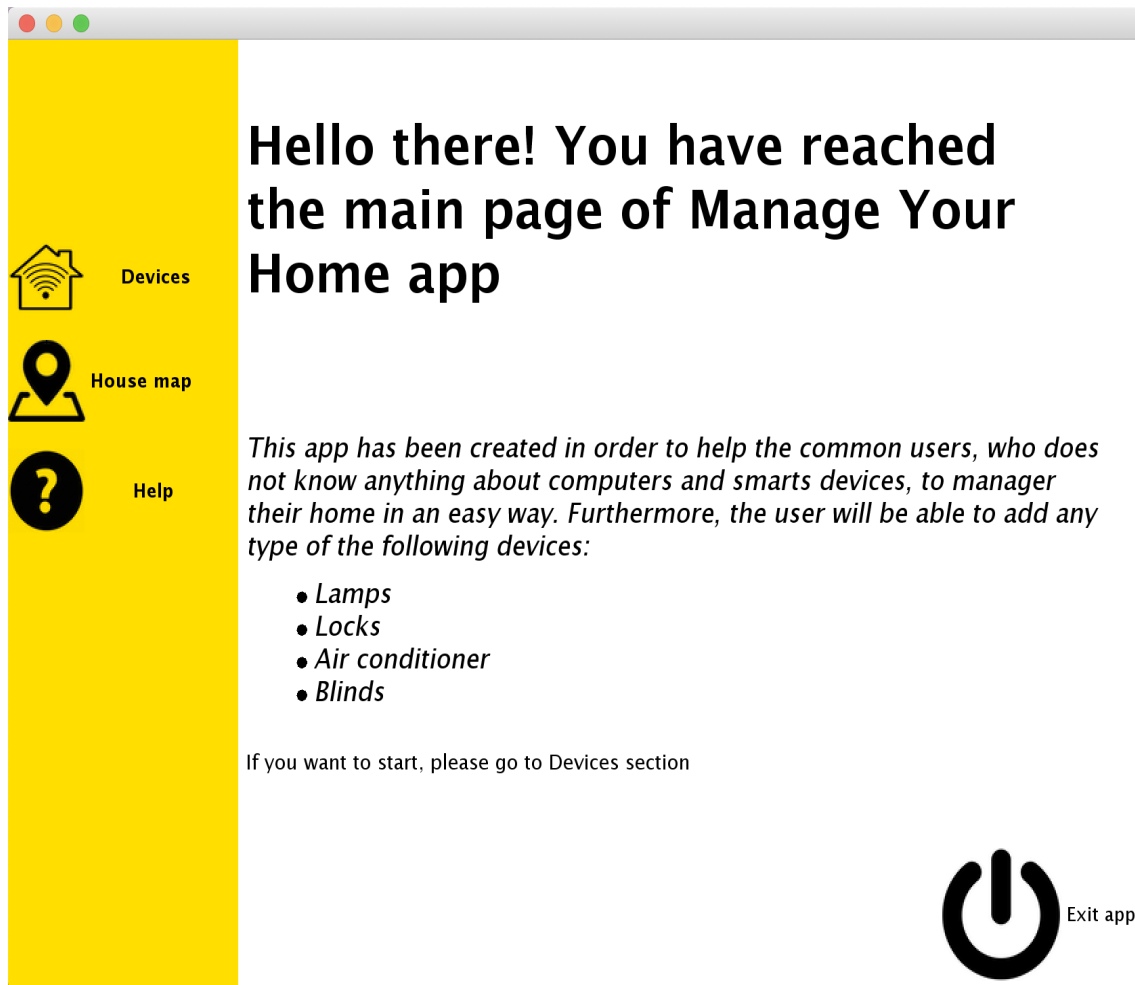


Figura 4: Página principal

3.3. Pantalla de dispositivos

En esta pantalla el usuario podrá observar los dispositivos inteligentes ya configurados en la aplicación. Como se puede observar en la Figura 5, el usuario podrá añadir cualquier dispositivo, editarlo y cambiar su estado de manera rápida y sencilla. Además, esta pantalla cuenta con un botón para eliminar cada uno de los dispositivos configurados.

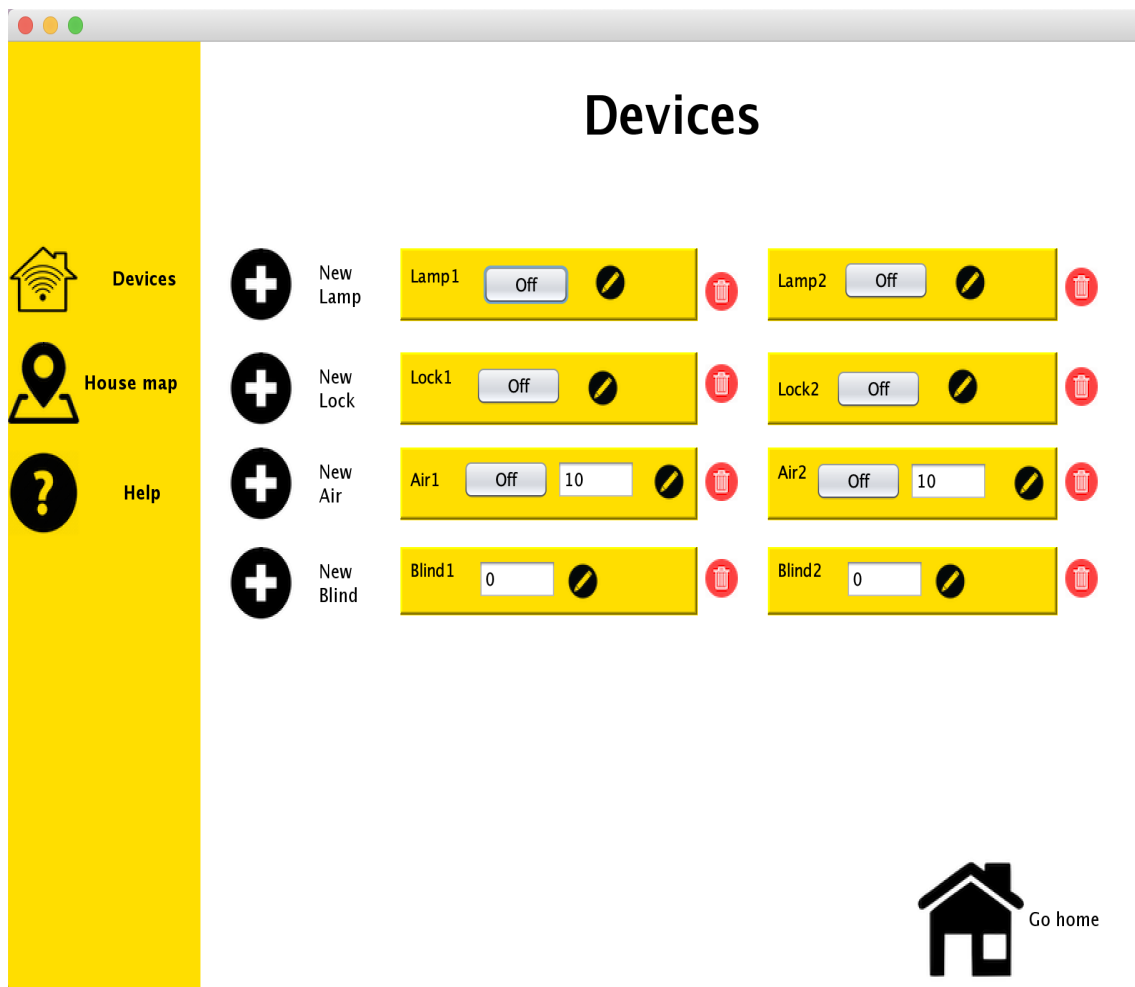


Figura 5: Página de dispositivos

3.4. Pantalla de nuevo dispositivo

Aunque para cada dispositivo tengamos que incluir una nueva variable, o eliminarla, todas las pantallas de nuevo dispositivo tienen el mismo estilo, ya que, para cada dispositivo necesitaremos un nombre y una dirección IP. En la Figura 6 se muestra la pantalla de nueva lámpara.

Como vemos, en el caso de las lámparas, podemos introducir el color de ésta. En cambio, para introducir un nuevo aire acondicionado, la sección de *Color* se cambia por una barra para seleccionar la temperatura a la que queremos que se encienda el dispositivo.

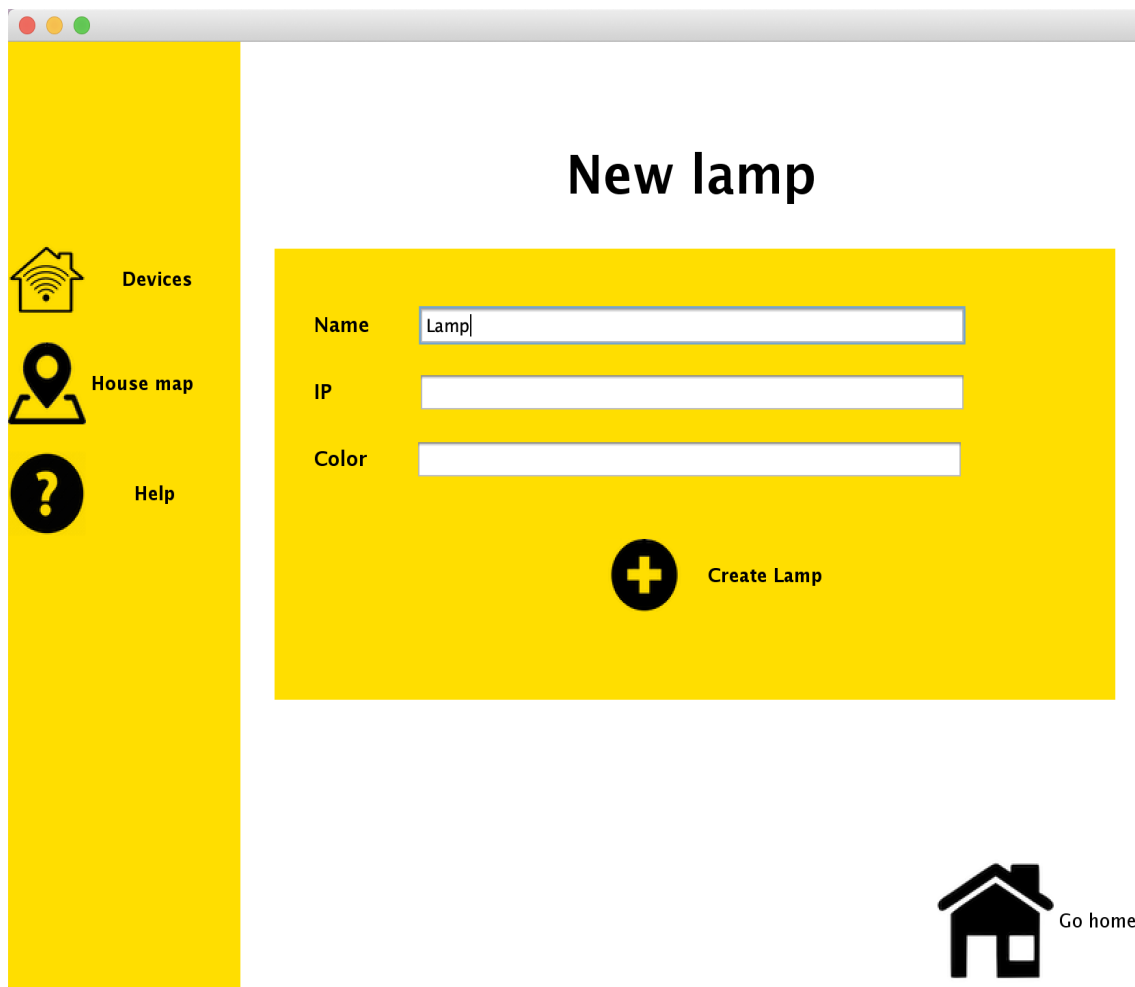


Figura 6: Página de nueva lámpara

3.5. Pantalla de edición

Para comprender mejor cómo está distribuida nuestra app, en esta sección mostraremos la pantalla de edición correspondiente a las lámparas. Como vemos en la Figura 7, el usuario no podrá modificar el nombre del dispositivo, pero sí la IP y el color del mismo.

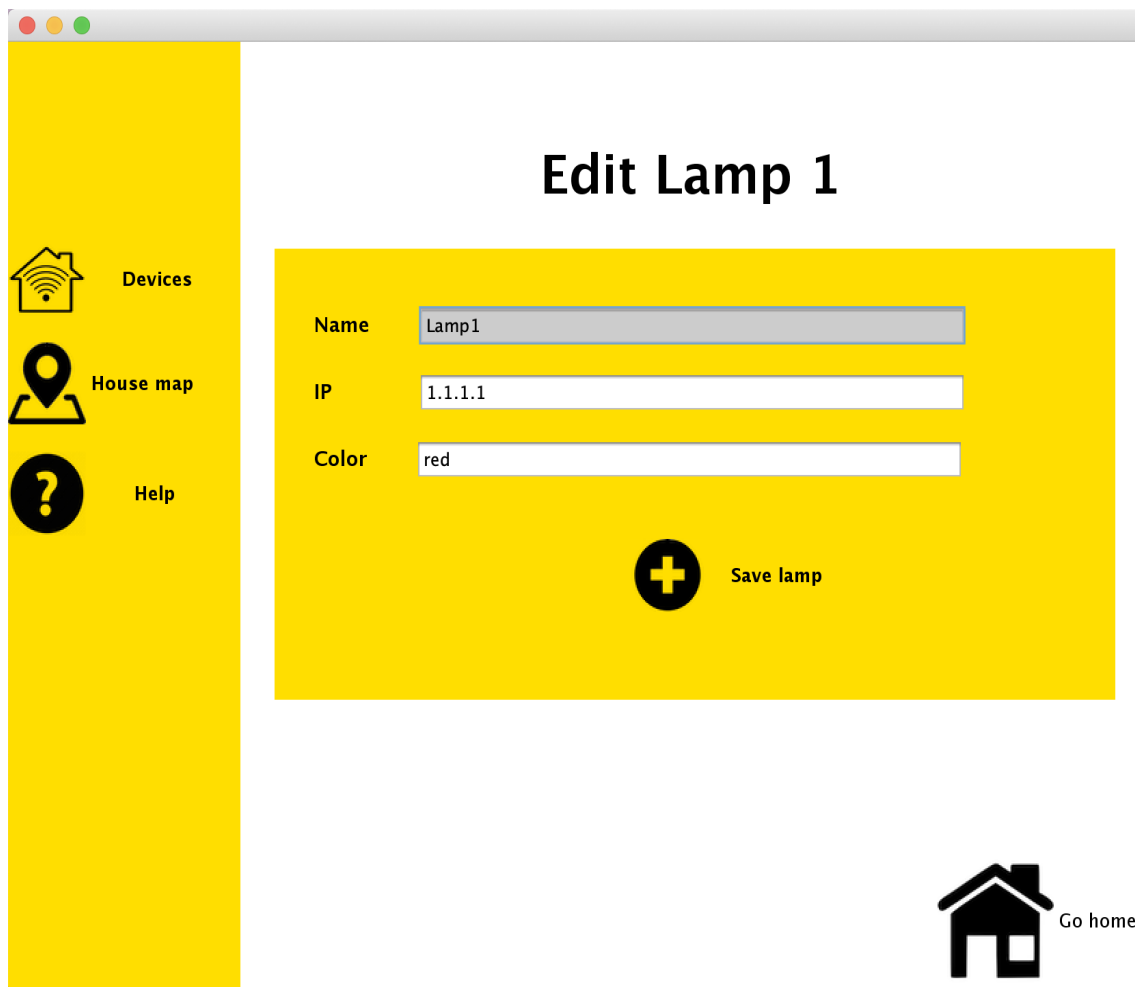


Figura 7: Página de editar lámpara

3.6. Pantalla Mapa

En esta pantalla el usuario podrá ver de manera visual y espacial sus dispositivos configurados y cuál es su estado actual. Como vemos en la Figura 8, los dispositivos actualmente apagados se encuentran en negro, en el caso de las lámparas, si la bombilla está en negro, significa que está apagada, en el caso de estar encendida, ésta no tendría el fondo negro. En el caso de los cerrojos, si éste está cerrado se encontrará con un cerrojo cerrado, en caso de estar abierto, se encontrará con un cerrojo abierto. Para el aire acondicionado, si éste se encuentra apagado el botón no aparecerá echando aire, en cambio, si está encendido el botón empezará a soltar aire. Por último, para las persianas, si éstas se encuentran cerradas, el botón aparecerá como persiana echada. En cambio, si está

abierta aunque sea un centímetro, este botón aparecerá como persiana subida.

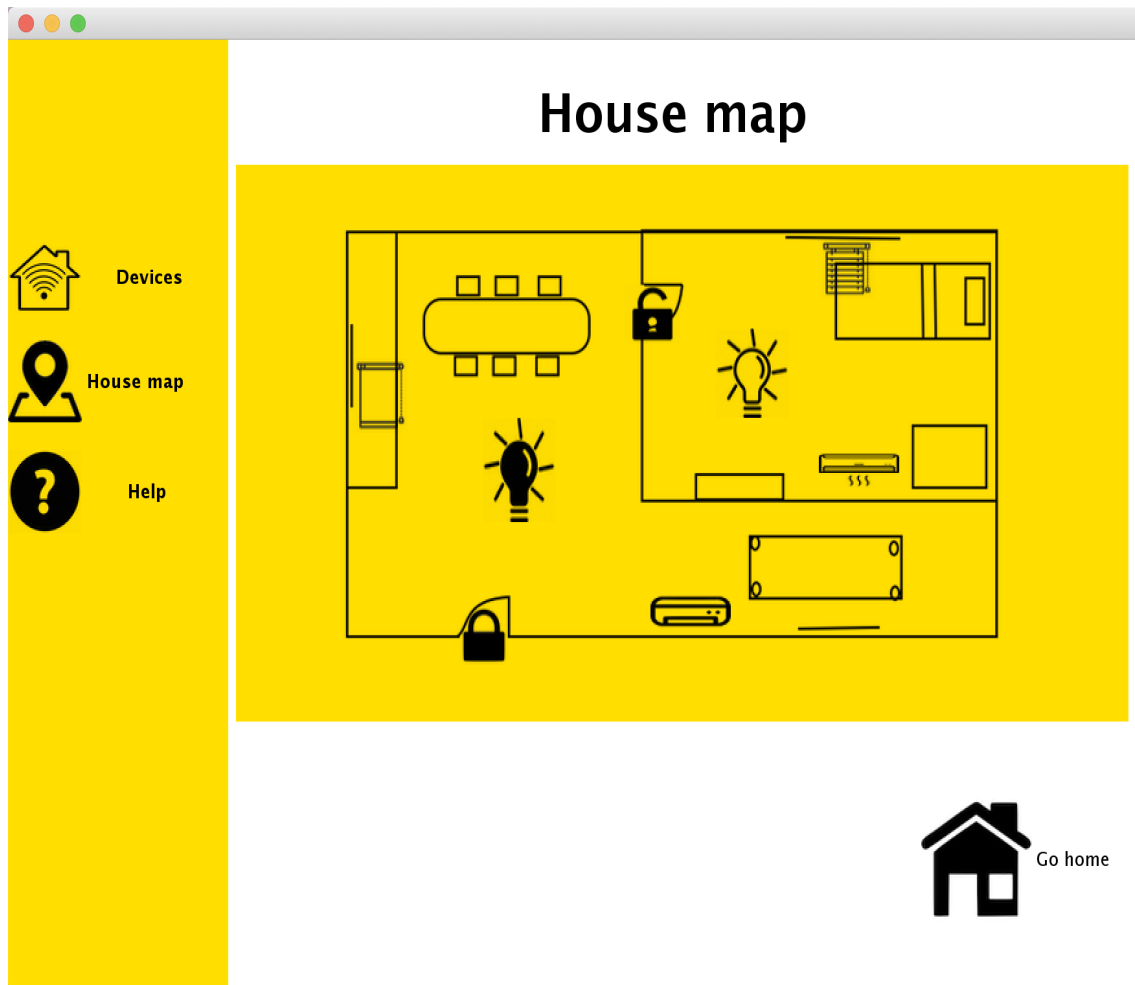


Figura 8: Pantalla Mapa

3.7. Pantalla de ayuda

En esta pantalla el usuario podrá ver cuáles son los pasos para utilizar esta aplicación. Además, en esta pantalla se muestra una breve explicación de cada una de las pantallas disponibles dentro de la aplicación. En la Figura 9 se puede observar esta pantalla.

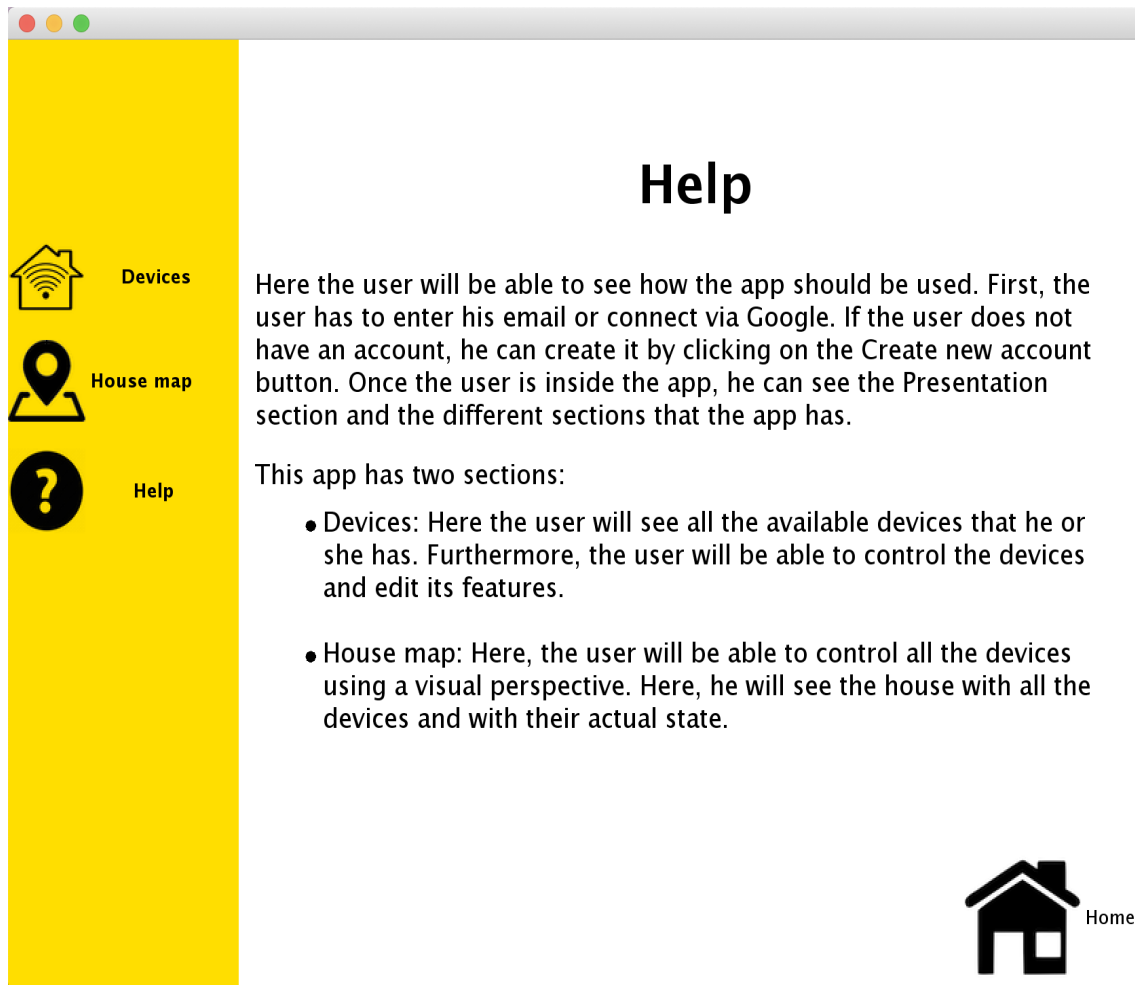


Figura 9: Página de ayuda

4. Implementación

En esta sección se explicará cómo hemos abordado la implementación de esta aplicación junto con la explicación de cada una de las clases creadas para su creación.

4.1. Clase *Lamp*

Esta clase consta de cuatro variables privadas las cuales representarán la IP, el nombre, el estado y el color de la lámpara. Además, en esta clase podremos encontrar los observadores y los modificadores de cada una de las variables privadas. Esta clase se utilizará para recoger la información de un fichero y guardarla en la variable para poder editar dicho fichero.

4.2. Clase *Lock*

Esta clase consta de tres variables privadas que indentifican la IP, el nombre y el estado del cerrojo. Al igual que la clase *Lamp*, esta clase cuenta con los observadores y modificadores de todas sus variables privadas. Esta clase se ha creado con la misma intención que la clase *Lamp*, es decir, para guardar la información de un fichero para poder así modificarlo posteriormente.

4.3. Clase *Air*

Esta clase ha sido creada para poder controlar la información de los ficheros correspondiente a este dispositivo. Para ello, hemos creado cuatro variables privadas que representan la IP, el nombre, el estado y la temperatura del aire acondicionado. La intención de esta clase es la misma que la de las dos clases anteriores.

4.4. Clase *Blind*

Al igual que la clase *Lock*, esta clase consta de tres variables privadas que identifican la IP, el nombre y el nivel de apertura de una persiana. Esta clase ha sido creada para poder editar los ficheros que guardan su información.

4.5. Clase Mail

Esta clase es la encargada de guardar en un fichero los usuarios creados hasta el momento. Esta clase se corresponde a la Figura 2. Para guardar el correo o nombre introducido por el usuario en un fichero, hemos hecho uso de las bibliotecas *File*, *FileReader* y *BufferedReader*. Lo que hacemos es leer todo el fichero con los emails, y, en el caso de que no exista, este será guardado en el fichero y el usuario será redireccionado a la página principal. En el caso de existir, la aplicación devolverá error y dará la posibilidad al usuario de volver a intentarlo.

4.6. Clases de nuevo dispositivo

Para crear un nuevo dispositivo hemos creado cuatro clases (*NewLamp*, *NewLock*, *NewAir*, *NewBlind*, en las cuales hemos creado una función como la que se muestra en el Listing 1 que guardará los valores introducidos por el usuario en un fichero que tendrá el nombre del dispositivo seguido del número de dispositivo. Ejemplo: *Lamp1.txt*.

```
1 void createFile(){
2     String src = "src/devices/" + nameTextField.getText() + ".txt";
3     File fout = new File(src);
4     FileOutputStream fos = null;
5     try {
6         fos = new FileOutputStream(fout);
7     } catch (FileNotFoundException ex) {
8         Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
9     }
10    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(fos));
11
12    // Add name
13    try {
14        bw.write(nameTextField.getText());
15    } catch (IOException ex) {
16        Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
17    }
18    // New line
19    try {
20        bw.newLine();
21    } catch (IOException ex) {
22        Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
23    }
24    // Add ip
25    try {
26        bw.write(ipTextField.getText());
27    } catch (IOException ex) {
28        Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
29    }
30    // New line
31    try {
32        bw.newLine();
```

```

33     } catch (IOException ex) {
34         Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
35     }
36
37     // Add state
38     try {
39         bw.write("Off");
40     } catch (IOException ex) {
41         Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
42     }
43     // New line
44     try {
45         bw.newLine();
46     } catch (IOException ex) {
47         Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
48     }
49
50     // Add state
51     try {
52         bw.write(colorTextField.getText());
53     } catch (IOException ex) {
54         Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
55     }
56     // New line
57     try {
58         bw.newLine();
59     } catch (IOException ex) {
60         Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
61     }
62
63     try {
64         bw.close();
65     } catch (IOException ex) {
66         Logger.getLogger(NewLamp.class.getName()).log(Level.SEVERE, null, ex);
67     }
68 }

```

Listing 1: Función createFile de la clase *NewLamp*

4.7. Clases de edición de dispositivo

Estas clases, dos por dispositivo (*editLamp1*, *editLamp2*, han sido creadas todas iguales. La diferencia entre estas clases es la variable del dispositivo que vayamos a cambiar. En el caso de una lámpara podremos editar su IP y su color, teniendo que crear una función que, primero lea el fichero actual y, segundo, cambiar ese valor por el nuevo introducido. Las funciones adicionales son las siguientes:

```

1  Lamp readFile() throws FileNotFoundException, IOException{
2      File f = new File("src/devices/Lamp1.txt");
3      String[] features = new String[20];
4      int i=0;
5      Lamp aux = new Lamp();
6
7      Scanner scanner = new Scanner(f);
8      while (scanner.hasNextLine()) {
9          String line = scanner.nextLine();
10         features[i] = line;

```

```

11         i++;
12     }
13
14     aux.setName(features[0]);
15     aux.setIP(features[1]);
16     aux.setColor(features[3]);
17
18     return aux;
19 }

```

Listing 2: Función readFile de la clase *editLamp1*

```

1 void changeIp(Lamp old, Lamp edit) throws IOException{
2     Path path = Paths.get("src/devices/Lamp1.txt");
3     Stream<String> lines = Files.lines(path);
4     List<String> replacedIp = lines.map(line -> line.replaceAll(old.getIp(),
5 edit.getIp())).collect(Collectors.toList());
6     Files.write(path, replacedIp);
7     lines.close();
8 }

```

Listing 3: Función changeIP de la clase *editLamp1*

```

1 void changeColor(Lamp old, Lamp edit) throws IOException{
2     Path path = Paths.get("src/devices/Lamp1.txt");
3     Stream<String> lines = Files.lines(path);
4     List<String> replacedColor = lines.map(line -> line.replaceAll(old.
5 getColor(), edit.getColor())).collect(Collectors.toList());
6     Files.write(path, replacedColor);
7     lines.close();
8 }

```

Listing 4: Función changeColor de la clase *editLamp1*

```

1 void resetComponents() throws IOException{
2     Lamp old = new Lamp();
3
4     old = readFile();
5     nameTextField.setText(old.getName());
6     ipTextField.setText(old.getIp());
7     colorTextField.setText(old.getColor());
8 }

```

Listing 5: Función resetComponents de la clase *editLamp1*

```

1 private void SaveButtonMouseReleased(java.awt.event.MouseEvent evt) { //GEN-
2 FIRST:event_SaveButtonMouseReleased
3     try {
4         // TODO add your handling code here:
5         Lamp aux = new Lamp();
6         Lamp old = new Lamp();
7
8         try {
9             old = readFile();
10
11             aux.setName(nameTextField.getText());
12             aux.setIP(ipTextField.getText());
13             aux.setColor(colorTextField.getText());
14
15             editFile(old, aux);
16         } catch (IOException ex) {
17             Logger.getLogger(editLamp1.class.getName()).log(Level.SEVERE, null,
18 ex);
19 }

```

```

18         }
19
20
21         Devices screen = new Devices();
22         screen.setVisible(true);
23         this.setVisible(false);
24     } catch (IOException ex) {
25         Logger.getLogger(editLamp1.class.getName()).log(Level.SEVERE, null, ex);
26     }
27 } //GEN-LAST:event_SaveButtonMouseReleased

```

Listing 6: Función SaveButtonMouseReleased de la clase *editLamp1*

4.8. Clase Devices

Esta clase es la que se encarga de mostrar por pantalla los datos relacionados con los dispositivos configurados en nuestro hogar. Para ello, vamos a pasar a explicar cómo mostramos los datos relacionados a los dispositivos *Lamp*, ya que los demás dispositivos se han creado de la misma manera. Lo primero que vamos a decir es que para las variables privadas de los dispositivos que sean de tipo entero, hemos creado dos variables privadas en esta clase para controlar el valor antiguo de ese dispositivo, para así poder realizar los cambios en tiempo real y de manera rápida. Con respecto a las funciones auxiliares utilizadas, la principal es la que se muestra en el Listing 7.

```

1  void initializeDevices() throws IOException{
2      if(checkFile("src/devices/Lamp1.txt") == true){
3          initializeLamp("Lamp1");
4      }else{
5          stateLamp1.setVisible(false);
6          editLamp1.setVisible(false);
7      }
8
9      if(checkFile("src/devices/Lamp2.txt") == true){
10         initializeLamp("Lamp2");
11     }else{
12         stateLamp2.setVisible(false);
13         editLamp2.setVisible(false);
14     }
15
16     if(checkFile("src/devices/Lock1.txt") == true){
17         initializeLock("Lock1");
18     }else{
19         stateLock1.setVisible(false);
20         editLock1.setVisible(false);
21     }
22
23     if(checkFile("src/devices/Lock2.txt") == true){
24         initializeLock("Lock2");
25     }else{
26         stateLock2.setVisible(false);
27         editLock2.setVisible(false);
28     }

```

```

29
30     if (checkFile("src/devices/Air1.txt") == true){
31         initializeAir("Air1");
32     }else{
33         stateAir1.setVisible(false);
34         editAir1.setVisible(false);
35         Air1TextField.setVisible(false);
36     }
37
38     if (checkFile("src/devices/Air2.txt") == true){
39         initializeAir("Air2");
40     }else{
41         stateAir2.setVisible(false);
42         editAir2.setVisible(false);
43         Air2TextField.setVisible(false);
44     }
45
46     if (checkFile("src/devices/Blind1.txt") == true){
47         initializeBlind("Blind1");
48     }else{
49         levelBlind1.setVisible(false);
50         editBlind1.setVisible(false);
51     }
52
53     if (checkFile("src/devices/Blind2.txt") == true){
54         initializeBlind("Blind2");
55     }else{
56         levelBlind2.setVisible(false);
57         editBlind2.setVisible(false);

```

Listing 7: Función initializeDevices de la clase *Devices*

Como vemos, esta función es la encargada de inicializar los valores. En el caso de que el fichero del dispositivo no exista, los componentes pertenecientes a ese dispositivo no se mostrarán por pantalla. Seguidamente, para inicializar estos valores en el caso de que exista el fichero, hemos creado la función que se muestra en el Listing 8.

```

1     void initializeLamp(String name){
2         File f = new File("src/devices");
3         int count = 0;
4         int linenum = 0;
5         File[] matchingFiles = f.listFiles(new FilenameFilter() {
6             public boolean accept(File dir, String name) {
7                 return name.startsWith(name) && name.endsWith(".txt");
8             }
9         });
10
11         String features[] = new String[4];
12         try {
13             FileReader fileReader = new FileReader(f+"/"+name+".txt");
14             BufferedReader bufferedReader = new BufferedReader(fileReader);
15             StringBuffer stringBuffer = new StringBuffer();
16             String line;
17
18             while ((line = bufferedReader.readLine()) != null && linenum < 4) {
19                 stringBuffer.append(line);
20                 features[linenum] = line;
21                 stringBuffer.append("\n");
22                 linenum++;
23             }
24             if (name == "Lamp1"){

```

```

25         fileRead1(features);
26     } else if(name == "Lamp2"){
27         fileRead2(features);
28     }
29
30
31     fileReader.close();
32 } catch (IOException e) {
33     e.printStackTrace();
34 }
35 }

```

Listing 8: Función initializeLamp de la clase *Devices*

Como vemos, esta función recogerá el valor de las variables guardadas en el fichero y se las pasará a otras funciones que se encargarán de mostrarla por pantalla. Estas funciones son las mostradas en el Listing 9.

```

1  void fileRead1(String[] features){
2      nameLamp1.setText(features[0]);
3      stateLamp1.setText(features[2]);
4  }
5
6  void fileRead2(String[] features){
7      nameLamp2.setText(features[0]);
8      stateLamp2.setText(features[2]);
9  }

```

Listing 9: Función fileRead1 y fileRead2 de la clase *Devices*

Las funciones explicadas hasta ahora, son las necesarias para inicializar todas las variables de la pantalla. Una vez inicializadas, si el usuario cambia el estado de una lámpara de encendido a apagado, esta acción se guardará en el fichero gracias a la función mostrada en el Listing 10.

```

1  void changeStateFile(String aux, String state){
2      try{
3          Path path = Paths.get(aux);
4          Stream<String> lines = Files.lines(path);
5          List<String> replaced = null;
6          if(state == "On"){
7              replaced = lines.map(line -> line.replaceAll("On", "Off")).collect(
Collectors.toList());
8          } else{
9              replaced = lines.map(line -> line.replaceAll("Off", "On")).collect(
Collectors.toList());
10         }
11         Files.write(path, replaced);
12         lines.close();
13     } catch (IOException e){
14         e.printStackTrace();
15     }
16 }

```

Listing 10: Función changeStateFile de la clase *Devices*

Por último, si el usuario deseara eliminar un dispositivo, bastaría con pulsar el botón eliminar y se llamaría a una función que eliminaría el

fichero de ese dispositivo. Esta función se muestra en el Listing 11.

```
1 void deleteFile(String path, String device){
2     try {
3         File file = new File(path);
4         Boolean exists = checkFile(path);
5         if(exists == true){
6             if(file.delete()){
7                 try {
8                     JOptionPane.showMessageDialog(null, device +" successfully
9 deleted. ", "Delete", JOptionPane.INFORMATION_MESSAGE);
10                    Devices screen = new Devices();
11                    screen.setVisible(true);
12                    this.setVisible(false);
13                } catch (IOException ex) {
14                    Logger.getLogger(Devices.class.getName()).log(Level.SEVERE,
15 null, ex);
16                }
17            }
18        } else {
19            JOptionPane.showMessageDialog(null, "Device does not exists. ", "
20 Error!", JOptionPane.INFORMATION_MESSAGE);
21        }
22    } catch (IOException ex) {
23        Logger.getLogger(Devices.class.getName()).log(Level.SEVERE, null, ex);
24    }
25 }
```

Listing 11: Función deleteFile de la clase *Devices*

4.9. Clase Map

Esta clase se encarga de mostrar visualmente los dispositivos que el usuario ha configurado anteriormente. Para inicializar los componentes de esta clase, hemos utilizado la función mostrada en el Listing 7. La diferencia de esta clase con la clase *Devices* es que a la hora de inicializar los componentes, nos fijamos en el valor del estado del dispositivo. Si el dispositivo no está configurado, el botón de dicho dispositivo no aparece en pantalla. Dependiendo del estado, hemos creado una serie funciones que cambian los iconos de los botones de los dispositivos dependiendo de si están encendidos o apagados. Como hasta ahora, vamos a mostrar la función que inicializa los dispositivos *Lamp*, ya que el resto de los dispositivos son inicializados de la misma manera. Esta función se puede observar en el Listing 12.

```
1 void initializeLamp(String device){
2     File f = new File("src/devices");
3     int count =0;
4     int linenum = 0;
5     File[] matchingFiles = f.listFiles(new FilenameFilter() {
6         public boolean accept(File dir, String name) {
7             return name.startsWith(name) && name.endsWith(".txt");
8         }
9     });
10 }
```

```

9         });
10
11         String features[] = new String[4];
12         try {
13             FileReader fileReader = new FileReader(f+"/"+device+".txt");
14             BufferedReader bufferedReader = new BufferedReader(fileReader);
15             StringBuffer stringBuffer = new StringBuffer();
16             String line;
17
18             while ((line = bufferedReader.readLine()) != null && linenum < 4) {
19                 stringBuffer.append(line);
20                 features[linenum] = line;
21                 stringBuffer.append("\n");
22                 linenum++;
23             }
24
25             if(device == "Lamp1"){
26                 if(features[2].equals("On")){
27                     ImageIcon ii = new ImageIcon(getClass().getResource("/images/
LampOn.png"));
28                     Image image = ii.getImage().getScaledInstance(Lamp1.getWidth(),
Lamp1.getHeight(), Image.SCALE_SMOOTH);
29                     ii = new ImageIcon(image);
30                     Lamp1.setIcon(ii);
31                 }
32                 if(features[2].equals("Off")){
33                     ImageIcon ii = new ImageIcon(getClass().getResource("/images/
lampOff.png"));
34                     Image image = ii.getImage().getScaledInstance(Lamp1.getWidth(),
Lamp1.getHeight(), Image.SCALE_SMOOTH);
35                     ii = new ImageIcon(image);
36                     Lamp1.setIcon(ii);
37                 }
38             }
39             if(device == "Lamp2"){
40                 if(features[2].equals("On")){
41                     ImageIcon ii = new ImageIcon(getClass().getResource("/images/
LampOn.png"));
42                     Image image = ii.getImage().getScaledInstance(Lamp2.getWidth(),
Lamp2.getHeight(), Image.SCALE_SMOOTH);
43                     ii = new ImageIcon(image);
44                     Lamp2.setIcon(ii);
45                 }
46                 if(features[2].equals("Off")){
47                     ImageIcon ii = new ImageIcon(getClass().getResource("/images/
lampOff.png"));
48                     Image image = ii.getImage().getScaledInstance(Lamp2.getWidth(),
Lamp2.getHeight(), Image.SCALE_SMOOTH);
49                     ii = new ImageIcon(image);
50                     Lamp2.setIcon(ii);
51                 }
52             }
53             fileReader.close();
54         } catch (IOException e) {
55             e.printStackTrace();
56         }
57     }

```

Listing 12: Función initializeLamp de la clase *Map*

Una vez inicializado los componentes de esta clase, hemos creado una función para que cuando el usuario presione sobre algún botón de alguno de los dispositivos configurados, éste cambie su estado de encendido a

apagado o viceversa. Para ello, hemos utilizado la función que se muestra en el Listing 13.

```
1 private void Lamp1MouseReleased(java.awt.event.MouseEvent evt) {//GEN-FIRST:  
2   event_Lamp1MouseReleased  
3     Boolean aux2 = null;  
4     String path = "src/devices/Lamp1.txt";  
5     try {  
6       // TODO add your handling code here:  
7       Lamp aux = readFile(path);  
8       aux2 = aux.getState();  
9  
10      if(aux.getState() == true){  
11        changeStateFile(path, "On");  
12      }  
13      if(aux.getState() == false){  
14        changeStateFile(path, "Off");  
15      }  
16    } catch (IOException ex) {  
17      Logger.getLogger(Map.class.getName()).log(Level.SEVERE, null, ex);  
18    }  
19    Map screen = new Map();  
20    screen.setVisible(true);  
21    this.setVisible(false);  
22  }//GEN-LAST:event_Lamp1MouseReleased
```

Listing 13: Función Lamp1MouseReleased de la clase *Map*

5. Futuras mejoras

Manage your Home es una aplicación que podría ayudar mucho a los usuarios no expertos a la hora de configurar los dispositivos inteligentes de su hogar. En el caso de esta aplicación, la configuración de los dispositivos se hace a través de su IP, lo que conlleva que el usuario debe de tener unos conocimientos mínimos al respecto. Como futura mejora, esta aplicación podría hacer como la de *Home*, creada por *Google*, que encuentra los dispositivos de manera automática. Además de esta posible mejora, hay algunas más que son destacables:

1. El uso de un almacenamiento en la red a través de bases de datos en vez de ficheros.
2. Posibilidad de crear más de dos dispositivos por categoría, haciendo que la interfaz sea de tipo *responsive*.
3. Posibilidad de clasificar los dispositivos en vez de por categoría, por habitación.
4. Poder entrar a la aplicación a través de la cuenta de *Google*, utilizando, por ejemplo, Auth0.
5. Posibilidad de crear más categorías de dispositivos inteligentes, ya sean televisores o instrumentos de cocina como las batidoras o cafeteras.
6. Añadir la posibilidad del control por voz para cada dispositivo inteligente. Para ello se incluiría un botón que, cuando es presionado, activa el reconocimiento por voz.

Referencias

- [1] <https://www.bosch-home.es/catalogo-electrodomesticos/cafeteras/cafeteras-totalmente-integrables/CTL636ES6>.
- [2] <https://www2.meethue.com/es-es/philips-hue-app>.
- [3] <https://github.com/RUGSoftEng/2018-Hestia-IOS>.