Timebox 7 – Scan fragment

Oversigt

OpgaveNavn	Implementering af scan fragmentet		
Implementering af krav	Delvis implementering af krav APP-5		
Udført af	Marc	**Dato**	05-11-2021
Timebox	7	**Område**	Mobilapp

Contents

INTRODUKTION **1**
ANALYSE **2**
DESIGN **2**
IMPLEMENTERING **5**
VERIFIKATION **12**
KONKLUSION **12**

Introduktion

Dette dokument omhandler implementeringen af fragmentet, der skal stå for at skanne andre Bluetooth enheder, hente brugerens adgangsinformationer, sammenligne dem mod de skannet enheder og give brugeren mulighed for at vælge en af de skannet enheder.

Analyse

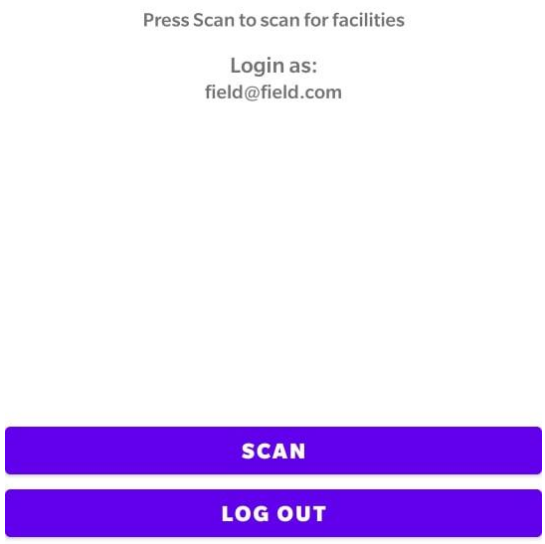
Det skal være muligt for bruger at:

- Se hvem man er logget ind som
- Der skal være tilstrækkelig Information om, hvad man skal gøre og hvad der er sket
- To knapper, en til at logge ud og en til at skanne.

Efter skanning skal det være muligt for brugeren at vælge et facilitet som bruger vil låse op.

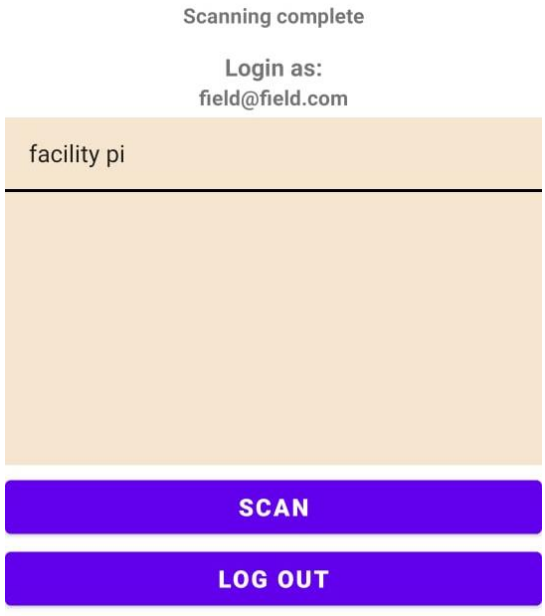
Design

Nedenstående Figur 1, viser layoutet efter man er logget ind.



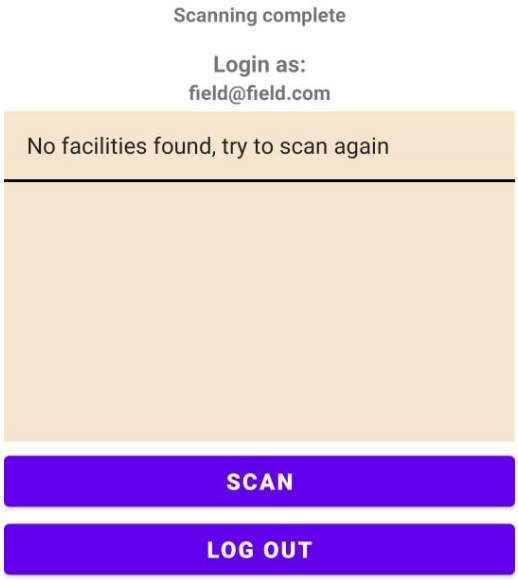
Figur 1 ScanFrag efter Login

En skanning med fundet facilitet, se på Figur 2.



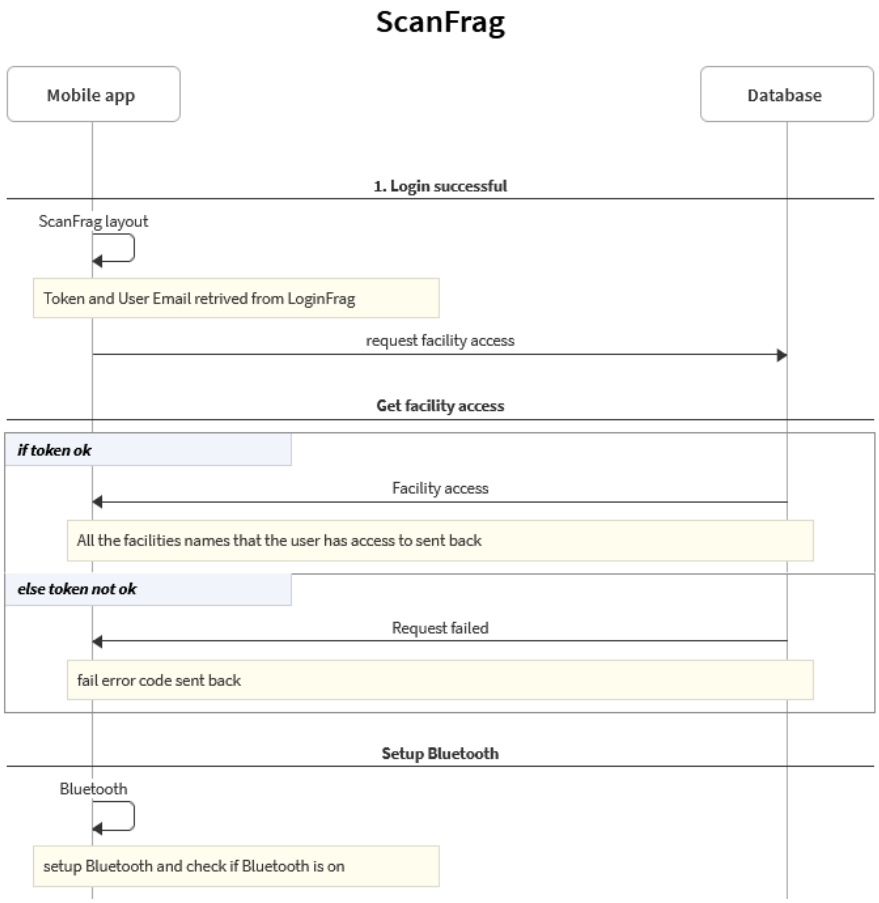
Figur 2 ScanFrag efter fundet enhed

En skanning uden fund af facilitet, se Figur 3.



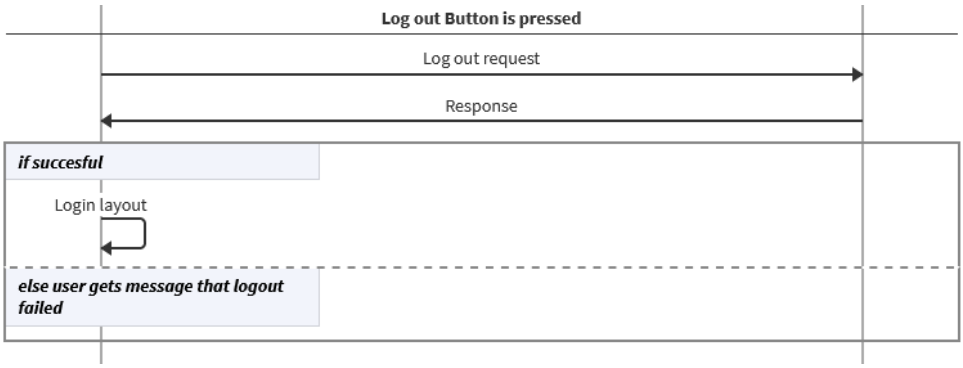
Figur 3 ScanFrag efter scan unden fund af enhed

Scan fragmentet forløber sig som på Figur 4, Figur 5 og Figur 6, hvor Figur 4 beskriver hvad der sker uden bruger indflydelse.



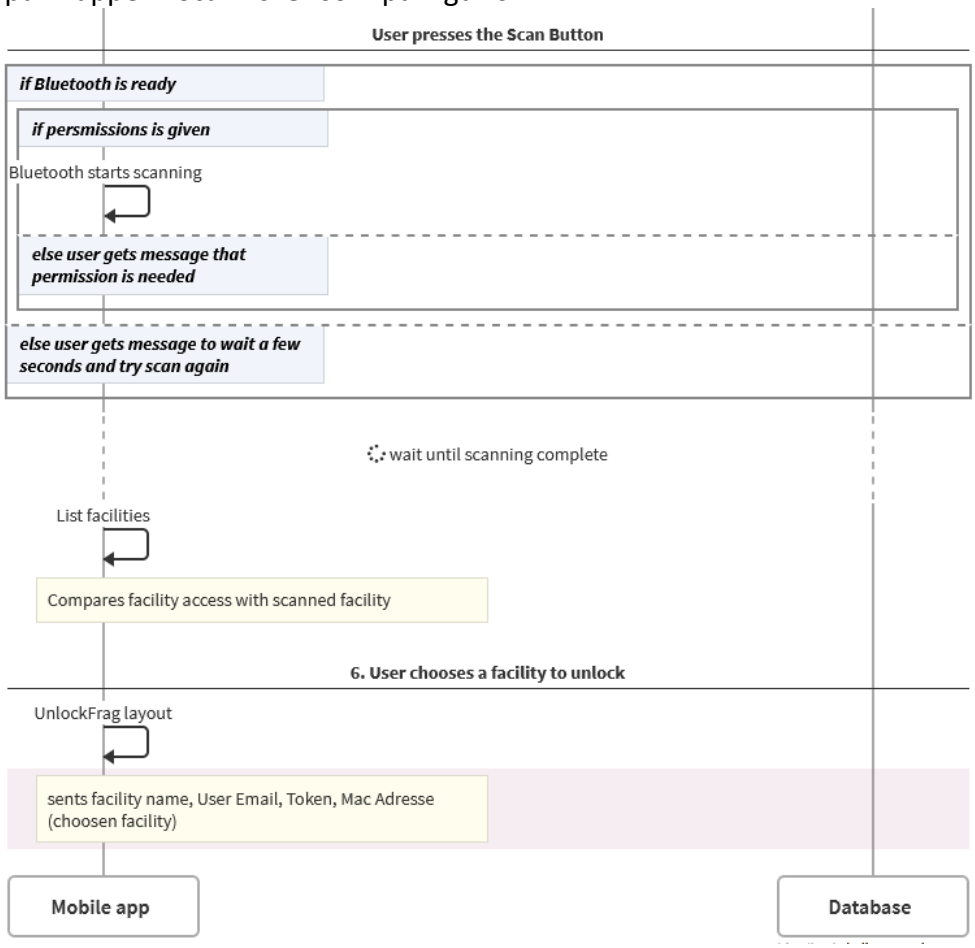
Figur 4 scanFrag del 1

Nedenstående Figur 5 viser aktiviteten for ”Log out” knappen.



Figur 5 ScanFrag del 2

Ved tryk på knappen ”Scan” sker som på Figur 6.



Figur 6 ScanFrag del 3

Implementering

Det første der sker i ScanFrag er at hente de argumenter (Token og UserEmail), se Figur 7, som bliver sendt med fra LoginFrag, hvor e-mailen bliver vist i et tekstfelt, så brugeren kan se at vedkommende er logget ind på den rigtige bruger.

```
65          //Get arguments
66          val args = this.arguments
67          //get a specific data entry in the bundle
68          val inputData = args?.get("UserEmail")
69          val inputToken = args?.get("Token")
70          //display bundle data
71          useremail = inputData.toString()
72          token = inputToken.toString()
73          val viewtest = view.findViewById<TextView>(R.id.textView)
74          viewtest.text = useremail
```

Figur 7 hent informationerne fra LoginFrag

Derefter hentes alle brugerens facilitetsadgang, se Figur 8, Figur 9, Figur 10 og Figur 11, sætter Bluetooth op og tjekker adgangen til Bluetooth, se Figur 12.

```
76          // Get facililty access
77          facilityAccess(useremail, inputToken.toString())
78          // Bluetooth
79          val bluetoothManager = context?.getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
80          mBluetoothAdapter = bluetoothManager.adapter
81          // Run Bluetooth checks
82          checkBT()
```

Figur 8 facility access, BT

Funktionen “facilityAccess” sætter Rest api op, klargører og sender beskeden som skal hente en list over brugerens facilitetsadgang, se Figur 9.

```
168     private fun facilityAccess(useremail: String, token: String){
169         //retrofit repo
170         val repository = Repository()
171         //retrofit modelFac
172         Log.d( tag: "tester", useremail)
173         Log.d( tag: "tester", token)
174         val viewModelFactory = FacInfoModelFactory(repository)
175         //model extension
176         viewModel = ViewModelProvider( owner: this, viewModelFactory).get(FacInfoViewModel::class.java)
177         //push POST to restAPI
178         val tokenPlace = "Token "
179         val tokenString = tokenPlace.plus(token)
180         val myPost = PostGetFacInfo(useremail, emptyList())
181         viewModel.pushPost(tokenString, myPost)
```

Figur 9 facilitetsadgang del 1

Derefter i Funktionen “facilityAccess” sættes der en observer op som lytter efter responset fra Databasen, som set på Figur 10, hvis responset er vellykket, ændres layoutet, hvorefter der tjekkes efter om brugeren har adgang til noget, hvis brugeren ikke har adgang til noget kan brugeren ikke skanne efter andre enheder og alle funktionalitet fjernes.

```
182         //read response
183         viewModel.myResponse.observe(viewLifecycleOwner, { response ->
184             Log.d( tag: "Iam IN", response.code().toString())
185             if (response.isSuccessful) {
186                 val progressBar: ProgressBar? = view?.findViewById(R.id.scanningBar)
187                 val loadingInfo = view?.findViewById<TextView>(R.id.loadingInfo)
188                 val listOfScanning = view?.findViewById<ListView>(R.id.scanResult)
189                 val buttonScan = view?.findViewById<Button>(R.id.btnScan)
190                 if(response.body()?.list.isNullOrEmpty()){
191                     Log.d( tag: "Response msg",  msg: "NO facility access")
192                     // Remove all features
193                     loadingInfo?.text = "You have NO access to any facility"
194                     // Make Visible
195                     progressBar?.visibility = View.INVISIBLE
196                     // Make Invisible
197                     listOfScanning?.visibility = View.INVISIBLE
198                     buttonScan?.visibility = View.INVISIBLE
```

Figur 10 facilitetsadgang del 2

Hvis brugeren har adgang til faciliteter, ændres layoutet så brugeren kan skanne efter enheder, se Figur 11.

```
199                 }else{
200
201                     // Remove all features
202                     loadingInfo?.text = "Press Scan to scan for facilities"
203                     // Make Invisible
204                     progressBar?.visibility = View.INVISIBLE
205                     listOfScanning?.visibility = View.INVISIBLE
206                     // Make Visible
207                     buttonScan?.visibility = View.VISIBLE
208                     Log.d( tag: "Response msg", response.message().toString())
209                     Log.d( tag: "Response code", response.code().toString())
210                     // get all facilities
211                     for(item in response.body()?.list!!){
212                         Log.d( tag: "Response string", item[0])
213                         facilityAccess.add(item[0])
214                     }
```

Figur 11 facilitetsadgang del 3

Funktionen ”checkBT”, se Figur 12, tjekker om:

- Mobilenheden har Bluetooth
- Bluetooth er aktiveret

```
226     private fun checkBT(){
227         if (mBluetoothAdapter == null) {
228             showToast( msg: "This device doesn't support Bluetooth")
229         }
230
231         // start BT if user has access to facilities
232         //make sure bluetooth is enabled.
233         if(!mBluetoothAdapter!!.isEnabled){
234             showToast( msg: "Bluetooth is OFF, trying to turn ON")
235             val enableBluetoothIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
236             resultContract.launch(enableBluetoothIntent)
237             showToast( msg: "Bluetooth is turned ON!")
238         }else{
239             if(mBluetoothAdapter!!.isEnabled){
240                 showToast( msg: "Bluetooth is already ON!")
241             }
242         }
243     }
244
245     // Checking Bluetooth
246     private val resultContract = registerForActivityResult(ActivityResultContracts.StartActivityForResult())
247     { result: ActivityResult? ->
248         if(result?.resultCode == Activity.RESULT_OK) {
249             showToast( msg: "Bluetooth has been enabled")
250         } else {
251             showToast( msg: "Bluetooth has been disabled")
252         }
253     }
```

Figur 12 Bluetooth tjek

Knappen ”Log out” er som på Figur 13.

```
84          // finds log out button
85          val btnLogOut = view.findViewById<Button>(R.id.btnLogOut)
86          btnLogOut.setOnClickListener {  it: View!
87              logout()
88          }
```

Figur 13 Log out button

Funktionen “logout” er som på Figur 14, bruger Rest Api, med en ”observer”, hvis svaret er ”successful”, vises der en besked, der fortæller brugeren ”logged Out” og omdirigere brugeren til ”LoginFrag”.

```
104    private fun logout(){
105        showToast( msg: "Log Out Click")
106        val repository = Repository()
107        //retrofit modelFac
108        val viewModelFactory = LogoutFragModelFactory(repository)
109        //model extension
110        viewModel2 = ViewModelProvider( owner: this, viewModelFactory).get(LogoutFragViewModel::class.java)
111        //push POST to restAPI
112        val tokenPlace = "Token "
113        val tokenString = tokenPlace.plus(token)
114        viewModel2.pushPost(tokenString)
115        //read response
116        viewModel2.myResponse.observe(viewLifecycleOwner, { response ->
117            if(response.isSuccessful) {
118                // Go back to login fragment
119                Log.d( tag: "Response", response.code().toString())
120                showToast( msg: "Logged out")
121                view?.let { Navigation.findNavController(it).navigate(R.id.action_scanFrag_to_loginFrag) }
122            }
```

Figur 14 Funktionen logout

Implementering af knappen “Scan” er som på Figur 15, hvor man kan se at brugeren får en besked om at skanning startes, derefter sker et tjek om Bluetooth er i gang med at skanne, hvorefter Bluetooth startes op.

```
90          // button for making new scans
91          val btnScan = view.findViewById<Button>(R.id.btnScan)
92          btnScan.setOnClickListener {  it: View!
93              showToast( msg: "Scanning started")
94              val loadingInfo = view?.findViewById<TextView>(R.id.loadingInfo)
95              if(!mBluetoothAdapter!!.isDiscovering) {
96                  loadingInfo?.text = "Scanning error wait X seconds"
97                  }
98              startBT()
99              }
```

Figur 15 Scan button

Funktionen ”startBT” er som på Figur 16, hvor der sker følgende:

- Der tjekkes om rettighederne givet
- Der tjekkes om Bluetooth er i gang med at skanne
- Bluetooth sættet til at skanne
- Der registreres en ”receiver” til at holde øje med Bluetooth enheden

```
132     private fun startBT() {
133         Log.d( tag: "sf_startBt",  msg: "function started")
134         val permissionCheck = ContextCompat.checkSelfPermission(
135             requireActivity(),
136             Manifest.permission.ACCESS_FINE_LOCATION
137         )
138         if (permissionCheck == PackageManager.PERMISSION_GRANTED){
139             Log.d( tag: "sf_startBt",  msg: "permissions granted")
140             // Register for broadcasts when a device is discovered.
141             val filter = IntentFilter().apply {  this: IntentFilter
142                 addAction(BluetoothDevice.ACTION_FOUND)
143                 addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED)
144                 addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED)
145             }
146
147             if (mBluetoothAdapter?.isDiscovering!!) {
148                 mBluetoothAdapter!!.cancelDiscovery()
149                 Log.d( tag: "test",  msg: "Canceling discovery.")
150                 mBluetoothAdapter!!.startDiscovery()
151                 // Register receiver
152                 requireActivity().registerReceiver(receiver, filter)
153             }else{
154                 Log.d( tag: "test",  msg: "Starting discovery.")
155                 mBluetoothAdapter!!.startDiscovery()
156                 // Register receiver
157                 requireActivity().registerReceiver(receiver, filter)
158             }
```

Figur 16 start Bluetooth

Bluetooth receiveren består er flere elementer, som reagerer på:

- ACTION_FOUND, som på Figur 17
- ACTION_DISCOVERY_STARTED, som på Figur 18
- ACTION_DISCOVERY _FINISHED, som på Figur 19 og en ”listener” som på Figur 20

”ACTION_FOUND” er implementere som på Figur 17, som gemmer alle skannet faciliteter, der har samme navn som brugeren har adgang til.

```
263         when(intent.action) {
264             BluetoothDevice.ACTION_FOUND -> {
265                 // Discovery has found a device. Get the BluetoothDevice
266                 // object and its info from the Intent.
267                 val device: BluetoothDevice? = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE)
268                 val deviceName = device?.name.toString()
269                 Log.d( tag: "deviceName",  msg: "" + deviceName)
270                 val deviceHardwareAddress = device?.address.toString() // MAC address
271                 Log.d( tag: "deviceHardwareAddress",  msg: "" + deviceHardwareAddress)
272                 // filter founded devices
273                 if(deviceName.contains( other: "facility")){
274                     // Get MAC address
275                     val deviceHardwareAddress = device?.address.toString()
276                     Log.d( tag: "deviceHardwareAddress",  msg: "" + deviceHardwareAddress)
277                     // Add  to list
278                     aLMac.add(deviceHardwareAddress)
279                     aLName.add(deviceName)
280                 } else {
281                     Log.d( tag: "action found",  msg: "discarded null named device")
282                 }
283             }
```

Figur 17 Bluetooth receiver ACTION_FOUND

“ACTION_ DISCOVERY_STARTED” som på Figur 18, sikre at den liste der gemmes de skannet faciliteter er tom, giver bruger en besked om at enheden er ved at skanne og en loading bar at kigge på.

```
284             BluetoothAdapter.ACTION_DISCOVERY_STARTED ->{
285                 // Make sure lists are cleared
286                 aLMac.clear()
287                 aLName.clear()
288                 loadingInfo?.text = "Scanning for facilities"
289                 // Make Visible
290                 progressBar?.visibility = View.VISIBLE
291                 // Make Invisible
292                 listOfScanning?.visibility = View.INVISIBLE
293                 buttonScan?.visibility = View.INVISIBLE
294             }
```

Figur 18 Bluetooth receiver DISCOVER_STARTED

"ACTION_DISCOVERY_FINISHED" er som på Figur 19 og Figur 20, hvor på Figur 19, tjekkes der om listen med fundet faciliteter er tom, hvis den er tom gives der en besked til bruger om der ingen faciliteter er fundet.

```
BluetoothAdapter.ACTION_DISCOVERY_FINISHED -> {
    Log.d( tag: "test",  msg: "in discovery finished receiver")
    loadingInfo?.text = "Scanning complete"
    // Make Invisible
    progressBar?.visibility = View.INVISIBLE
    // Make Visible
    listOfScanning?.visibility = View.VISIBLE
    buttonScan?.visibility = View.VISIBLE

    if(aLMac.isEmpty()){
        // No facilities founded
        Log.d( tag: "LIST EMPTY: ",  msg: "EMPTY MAC list")
        // listView setup with NO facilities founded
        val noneFound = ArrayList<String>()
        noneFound.add("No facilities found, try to scan again")
        val adapter = context.let { ArrayAdapter(it, android.R.layout.simple_list_item_1, noneFound)}
        val selectDeviceList: ListView? = view?.findViewById(R.id.scanResult)
        selectDeviceList?.adapter = adapter
        Log.d( tag: "Bundle: ",  msg: "username: " + useremail)
        Log.d( tag: "Bundle: ",  msg: "token: " + token)
```

Figur 19 Bluetooth receiver DISCOVERY_FINISHED

Hvis den skannede liste ikke er tom, sættes der en "ListView" op med de fundne faciliter. Der sættes så en "listener" op, som på Figur 20, som tager bruger input og sender Mac addressen, Token, User Email og facilitetsnavnet med videre til UnlockFrag.

```
// listView setup
val adapter = context.let { ArrayAdapter(it, android.R.layout.simple_list_item_1, showList)}
val selectDeviceList: ListView? = view?.findViewById(R.id.scanResult)
selectDeviceList?.adapter = adapter
selectDeviceList?.setOnItemClickListener{_,_,position,_ ->
    chosenDeviceName = showList[position]
    Log.d( tag: "arguments ff: ",  msg: "CDN: " + chosenDeviceName)
    chosenDeviceMac = MacList[position]
    Log.d( tag: "arguments ff: ",  msg: "CDM: " + chosenDeviceMac)
    // make bundle
    val bundle = Bundle()
    bundle.putString("UserEmail", useremail)
    Log.d( tag: "Bundle: ",  msg: "username: $useremail")
    bundle.putString("Token", token)
    Log.d( tag: "Bundle: ",  msg: "token: $token")
    bundle.putString("MacAdress", chosenDeviceMac)
    bundle.putString("FacName", chosenDeviceName)
    view?.let { Navigation.findNavController(it).navigate(R.id.action_scanFrag_to_unlockFrag, bundle) }
}
```

Figur 20 Bluetooth receiver listener

For at sikre at "receiveren" bliver "unregisteret", gøres det i "onDestroy" som på Figur 21.

```
override fun onDestroy() {
    try {
        if (receiver != null) {
            activity?.unregisterReceiver(receiver)
        }
    } catch (e: IllegalArgumentException) {
        e.printStackTrace()
    }
    super.onDestroy()
}
}
```

Figur 21 onDestroy

Verifikation

Det antages at:

- Nyeste version af mobilappen er startet op
- Test setup'et med embedded (med navn facility pi) er sat op
- Hjemmesiden med database er oppe og kører på AWS-serveren
- Man har adgang og logget ind på brugeren field@field.com
- Man har adgang til brugeren office@office.com

Tabel 1: Tests til verifikation af opgave

Test	Test Steps	Pass-betingelser	Resultat
Scan med field	1. Klik på "Scan" 2. Klik på "facility pi"	1. Man finder facility pi 2. efter man har trykket på facility pi kommer man en til næste layout (UnlockFrag)	Bestået
Scan uden at finde enheder	1. Klik på "Scan"	1. Man finder ingen faciliteter og får beskeden " No facilities found, try to scan again"	Bestået
Log ud	1. Klik på "Log out"	1. Man logger ud og kommer til LoginFrag layout	Bestået
Scan med office	1. Det er kun muligt at trykke på log out	1. Man kan ikke trykke på "Scan" og får beskeden "You have NO acces to any facility"	Bestået

Konklusion

Skanningsfragmentet er blevet implementeret, så det er muligt at skanne efter faciliter og sammenligne dem med brugerens adgang. Skanningsfragmentet sætter Bluetooth enheden op til at starte og stoppe med at skanne.
Det er muligt at vælge en fundet facilitet, som ønskes låst op.
Der er implementeret nogle bruger venlige beskeder til at guide brugeren i korrekt brug.