ALS & ALSSUND

This is the third chapter of the description detailing a card- and cashless system for Spejdernes Lejr 2017 – obviously in English. The first chapter focused on the opportunities provided by this solution. The second chapter was about cost and UI. This chapter chews on the technologies used.

Placing the product in the open-source domain with a GNU General Public License attached to it will allow developers to utilise it to their own projects in return for partaking in the crowd-sourcing effort.

open-source is another/fancy way of saying that everybody owns the source, or no one if you like!

crowd-sourcing is like when the Amish builds a barn! Everybody joins in and lend a helping hand.

Pieces to a puzzle

There are 2 main pieces and a small number of minor pieces to the puzzle. The two main pieces are: ALS og ALSSUND. An intelligent off-line first terminal (ALS) which is an app you may download to any Android smartphone, and a server (ALSSUND) which is an application server or a swarm of application servers, an Nginx load-balancing proxy web server, and a MySQL database server, all in one Docker composed set of containers.

ALS is worth nothing without the wristbands! They are a minor piece, being manufactured to specs provided by KOM. Part of the terminals will even attach a barcode scanner via USB – another minor piece, bought of-the-shelf.

ALSSUND will require a few resources and offers a set of RESTful API's for integrating these resources – a set of product and user attributes and *bank accounts*.

Artefact glossary

This problem (and solution) domain has a small number of artefacts explained below:

ALS

ALSSUND

Android

API

Docker

KOM

localStorage

MySQL

Nginx

normalised

Object

Process

RESTful

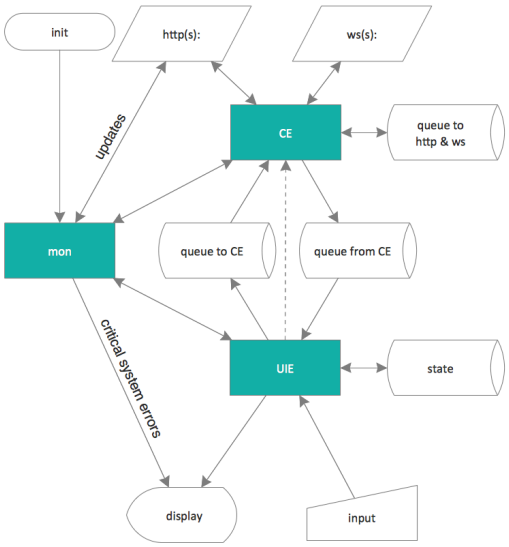
strike-3 watch list

Thread

Widget

ALS

The ALS app is an off-line first Android client with these basic building blocks:



Android will read the manifest.xml and spawn the mon process. The system **mon**itor process will check for updates and if no updates exist, it will spawn a **C**ommunication **E**ngine (CE) process and then spawn a **U**ser **I**nterface **E**ngine (UIE) process, handing over the CE object doing so. The UIE will open the queue to CE and announce it to CE and ask CE for a handle to the queue from CE, and open/establish a localStorage to persist state, and finally start drawing the initial window on the display.

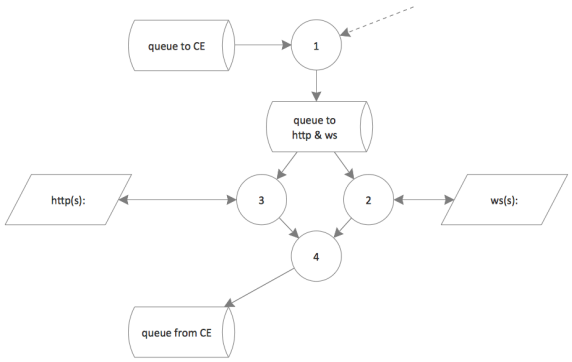
System Monitor

The system monitor has 4 tasks -

1) handle installing updates (or downdates) on start, and
2) keep an eye on the performance of the CE and UIE processes, starting them in the event that one or the other dies, and
3) tell the user of critical system errors that will require it to give in and exit, and
4) close down the app and processes once told to

Communication Engine

Modern dialogue is asynchronous, threaded and executed in parallel. The Communication Engine (CE) caters to this principle.



This process has one main thread whose task is to fork threads and report back to the monitor when questioned.

One thread handles

- reading messages off of the queue to CE
- immediately adds them to its own queue prioritising them in the process
- returns a status when asked

Another thread handles the socket and secure socket communication by

- unshifting messages, destined for socket(s), from the queue
- publishing the message on the socket in question
- pushing the message onto the queue if publishing failed
- sends messages from the sockets to the fourth thread
- returning a status when asked

A third thread handles the http and secure http communication by

- unshifting messages, destined for http(s), from the queue
- making the request in question
- pushing the message onto the queue if request failed
- sends responses from the network to the fourth thread
- returning a status when asked

A fourth thread handles

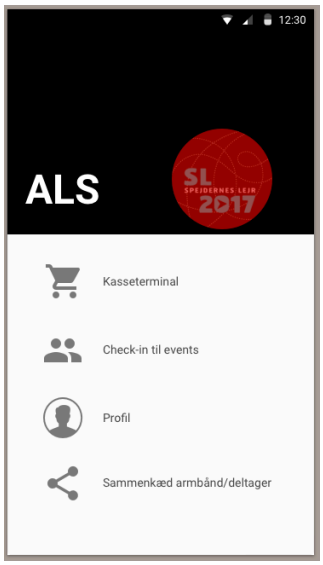
- setting up the connection to the queue (created by the UIE process)
- writing messages to the queue from CE
- returning a status when asked

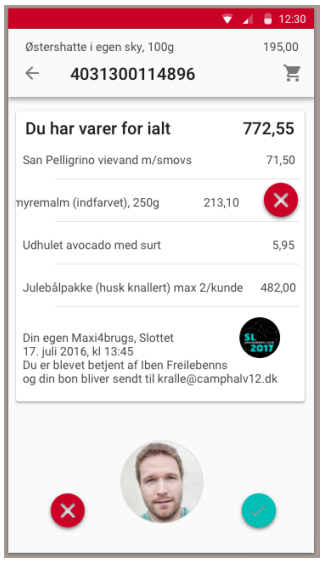
User Interface Engine

The UIE sets up any window and widget artefacts required by other processes, and starts by creating a queue to CE which it expects to be able to write to, then call the CE process to get the handle of the queue from which it expects to be able to read from.

It will setup a localStorage (or resume an existing one). All objects and all of state will be persisted to this localStorage.

Finally the UIE will display the current page as specified by state – or the landing page in case no state is set.


12:30
ALS
Kasseterminal
Check-in til events
Profil
Sammenkæd armbånd/deltager


12:30
Østershatte i egen sky, 100g 195,00
4031300114896
Du har varer for ialt 772,55
San Pelligrino vievand m/smovs 71,50
myremalm (indfarvet), 250g 213,10
Udhulet avocado med surt 5,95
Julebålpakke (husk knallert) max 2/kunde 482,00
Din egen Maxi4brugs, Slottet
17. juli 2016, kl 13:45
Du er blevet betjent af Iben Freilebenns
og din bon bliver sendt til kralle@camphalv12.dk

managing terminals

how do we counter MAC spoofing?

How To find MAC Address of Android Mobile?

- Unlock your android mobile and then move to menu options
- Find the settings in all the apps and goto Settings
- After moving in to setting scroll down there it appears About Phone
- Then goto About Phone and click on Hardware Information
- There you find Wi-Fi MAC address with a format like : 00:00:00:00:00:00
- If you don’t find the Hardware Information in your settings click on status
- After clicking on scroll down there it appears Wi-Fi MAC address

ALSSUND

There are a number of containers comprising ALSSUND. For one there is the application server providing the endpoints utilised by the ALS terminals. Secondly there is the load-balancing Nginx web server handling both static resources, dynamic resources and the multiplexing of web sockets, https://www.nginx.com/blog/realtime-applications-nginx/ and finally there is the MySQL RDBMS server which offers persistency.

Products, Users and Bank accounts

The external resources required by ALSSUND are the product and user objects, and the bank account objects.

The Product Object

Of the core provisions by ALS is being a POS terminal. To upkeep its promises on this issue ALS requires a product object on every item that ALS is supposed to be able to register as a POS. Every object must provide at the very least a name and a price but most will also provide multiple barcode numbers; an EAN13 number or a PLU or perhaps just 2-3 digits identifying the product.

The User Object

The other core provision by ALS is the authentication of users, and persistence of presence. The user object will provide at the very least, the users first name, a number and a portrait photo depicting the user facing the camera. Both provisions are offered utilising the NFC reader in Android smartphones on wristbands carrying a read/write RFID chip. ALSSUND holds all wristbands handed out and there are 3 attributes to each valid wristband; the RFID number, an 8 character code unique to each wristband, and the users number.

The Bank Account Object

It is the user's discretion to utilise the wristband as a cashless payment method. In the camp SL2017 the user will be required to do so! The bank account object will hold at least the users number, a balance, and the next salt key (more on this later).

MySQL Database

The information persisted by the MySQL database, are normalised into a number of tables. Primarily *wristbands* that persists information pertaining to the wristbands, and secondly *presences* which details where a wristband have been registered, and *terminals* that hold any terminal being used against ALSSUND, and finally *transactions* that are the point-of-sale transactions.

campOS integration

Den enkelte lejrdeltager har et nummer, og dette nummer er nøglen til brugerens oplysninger. ALSSUND skal kun bruge ganske få oplysninger, som den henter ved at lave et REST opslag mod campOS på formen: /api/v1/participants/:number og forventer at få et JSON objekt retur med de nødvendige oplysninger. Billedet af den enkelte deltager er et åbent spørgsmål. Findes det ikke her, kan det gemmes på Amazon S3.

Bank integrationen

Den enkleste 'bank' at integrere til, er en web(bank)shop, hvor ejere af armbånd (og deres velmenende familier/venner) kan 'købe' penge, og lægge dem på hylden. Her retter ALSSUND et REST opslag på formen: /api/v1/participants/:number og forventer at få et JSON objekt retur med de indbetalinger, der har været på det deltager nummer.

Viser det sig ikke muligt at etablere en sådan, er plan B at tilbyde en dosmerseddel, hvor den enkelte trop/gruppe kan huske, hvem der har betalt lommepenge ind til deres egen Swipp eller MobilePay, og når troppen/gruppen så overfører beløbet der står på dosmerseddelen, bliver de pågældende armbånd 'tanket op' automatisk, og ALSSUND kan så igen stille sin forespørgsel fra før.

Workflows

The following pages will detail each workflow, from initialisation of a terminal, to decommissioning that very same terminal. For reference the corresponding UI element is depicted where applicable.

Opening a socket to ALSSUND is short for doing a name lookup for alssund.sl2017.dk and sending the necessary HTTP(S) upgrade request in order to establish a web socket connection with alssund.sl2017.dk

Security considerations

The primary consideration is how to trust the DNS lookup, and hence actually be *talking* to ALSSUND. Therefore the actual name used is not disclosed!

Transmissions will be encrypted over wss (secure web sockets) and considered secure and thus should avert any man-in-the-middle attacks.

Another primary consideration is how to trust the terminal – it might have fallen into an ill-doers hands!

First an foremost: the terminal has to be approved for the transaction type (eg. POS terminal) **and** the host has to be approved for the transaction type as well!

But even so users will eventually loose their wristbands and such wristbands might end up in the hands of black-hats!

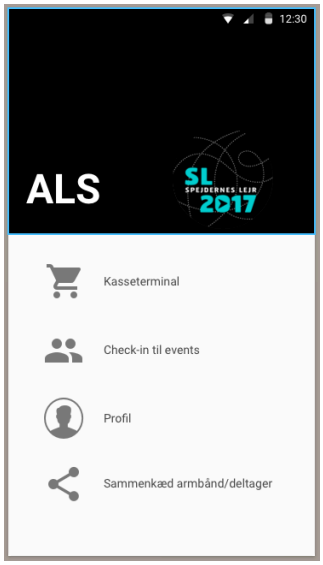
The following use cases with black-hats/ill-doers exist:

1) the ill-doer checks someone onto the camp area. However grim this action is, no real threat exist and no funds are lost.
2) the ill-doer links a ‘fake’ wristband to a user. With all wristbands registered prior to the SL2017 event, this use case is irrelevant!
3) the ill-doer links a wristband to a ‘fake’ user. Likewise all users are registered.
4) the ill-doer links a wristband to a user – but keeps the wristband to himself. This use case will be considered an act of simple theft and camp management will have to consider reporting it to the police. As soon as the user receives an email with the acknowledgement of his wristband being linked to his profile, and left without a wristband, he should report the ‘loss’ to camp management!
5) the ill-doer receives funds meant for a users profile credit build-up but keeps the funds to herself. A balance deficit between the bank account and user profiles credit sum will exist and alert accounting that an act of foul play is unfolding
6) the ill-doer receives funds meant for a users profile credit build-up but changes the amount being credited. As soon as the user receives an email with the acknowledgement of the credit being built, she will be able to recognise the discrepancy and should report it to camp management
7) the ill-doer skips scanning products (that’s called shop lifting)
8) the ill-doer skips validating the goods purchased and aborts the buying workflow. Doing this should be logged - and if the same host is seen doing this repeatedly an investigation should be the correct response
9) the ill-doer uses someone else’s wristband to validate a purchase. This is like stealing the purse or goods from another shopper and should be reported to camp management by the offended

The ALS Initialisation Workflow

On the first open of the ALS app on a terminal, ALS will identify that it does not have a terminal ID, hence ALS will build a public shared key set, open a socket to ALSSUND, a publish

`"ALS:PUBLICKEY:FF3E29BF2910CC0459AD"`



ALSSUND will publish

`"ALSSUND:PUBLICKEY:A74528FDA783CD4598FDA"`

ALS will hash its own public key, encrypt it with its own private key, encrypt that value with ALSSUND's public key and publish

`"ALS:NEW:FF3E29BF2910CC0459AD"`

ALSSUND will decrypt the third value with its own private key part and decrypt the result with ALS' public key, and de-hashing it should leave ALSSUND with ALS' public key, and 1 in 2 use cases to consider at this point in time:

Either terminal management will have added the MAC address to the terminals table with the role appointed to this terminal, and ALSSUND will take a note of the current IP address, and the public key part, and the terminal can continue initialisation,

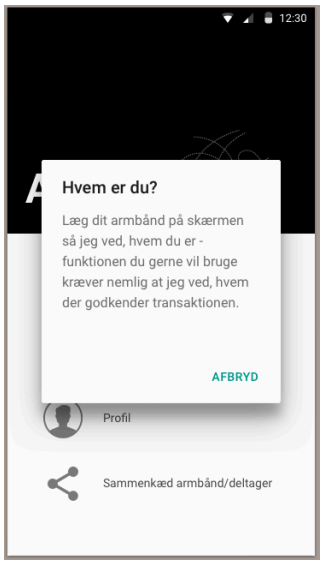
`"ALS:42:FF3E29BF2910CC0459AD"`

Or the MAC address is not in the terminals table, in which case ALSSUND will publish

`"ALS:IDENTIFY:HOST"`

ALS will ask the host (user) to identify herself. ALS will publish the result as

`"HOST:LOGON:A9BF2910CC0459AD"`



ALSSUND will validate the host and create a new terminal object with the MAC address and the current IP address and add the approve wristband role (S) only, add the third value as the terminal's public key, generate and encrypt a password with it, and publish

`"ALS:42:FF3E29BF2910CC0459AD"`

In case the terminal does exist already, could not be inserted, or some other error surfaced, ALSSUND will respond with

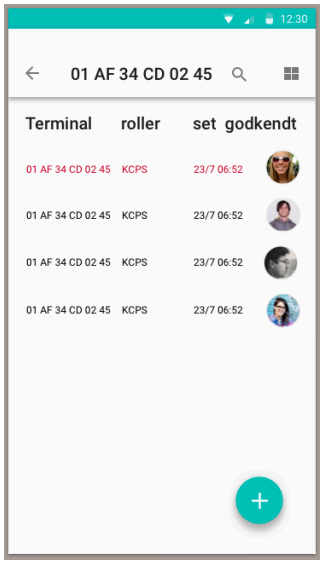
`"ALS:-1:REASON"`

The Terminal Management Workflow

Some terminals will be pre approved to do various tasks like acting as a POS terminal, build credit on profiles, work the check-in counters, etc

In order to manage the terminals you will need a special TERMINAL WRISTBAND – and don't loose it; there are but a select few!

Press the SL2017 logo and present your wristband.



ALS will publish

`"TERMINAL:ACCESS:FF3E29BF2910CC0459AD"`

ALSSUND will validate the wristband and publish

`"TERMINAL:ACCESS:GRANTED"`

ALS will publish

`"TERMINAL:LIST"`

ALSSUND will build a JSON object and publish a tmp file reference

`"TERMINAL:LISTED:/tmp/f3tymmewr2serga.enc"`

This object looks like this

```
{ terminals: [
      terminal: {
            timestamp: "23/7/17 14:55",
            mac: "01 AF 34 CD 02 45",
            roles: "CS"
            host: "/images/hosts/portrait-424235-xs.png"
      },
      . . .,
      terminal: {},
] }
```

ALS will GET and decrypt the JSON object and display it.

The user can press the 'plus' and add a terminal - and insert it presenting her own wristband

ALS will publish

`"TERMINAL:NEW:01AF34CD0245:CS"`

ALSSUND will insert the terminal and publish

`"TERMINAL:42:FF3E29BF2910CC0459AD"`

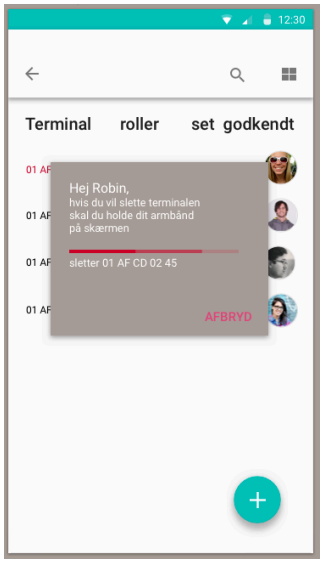
The user can swipe any terminal line left to delete it and approve of it presenting his own wristband.

ALS will publish

`“TERMINAL:42:DELETE”`

ALSSUND will delete the terminal and publish

`“TERMINAL:42:DELETED”`



If ALSSUND has an open connection with terminal 42, it will close the socket

The user can swipe any terminal line right to edit it and once done, approve of it presenting his own wristband.

ALS will publish

`“TERMINAL:42:EDIT:01AF34CD0245:CPS”`

ALSSUND will edit the terminal and publish

`“TERMINAL:42:EDITED”`

The ALS logon Workflow

ALS will open a socket to ALSSUND and must publish

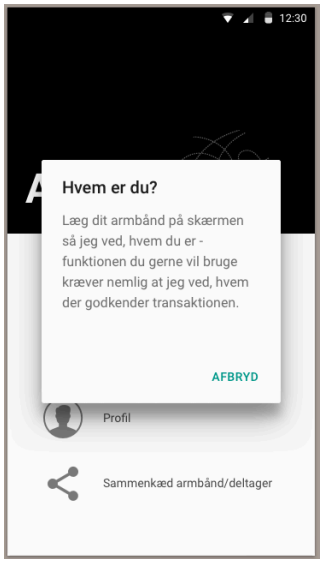
`"ALS:42:A9BF2910CC0459AD"`

ALSSUND will decrypt the third value using ALS #42's public key and validate that the third value is indeed the password used when ALS #42 was initialised. If the value does not match, the socket is closed and the from IP address and from MAC address are added to a strike-3 watch list.

Otherwise ALS is ready

The user will access any of the menu items. If a menu item requires the user to identify herself, a logon modal will appear and the user will present his wristband and ALS will publish

`"HOST:LOGON:A9BF2910CC0459AD"`

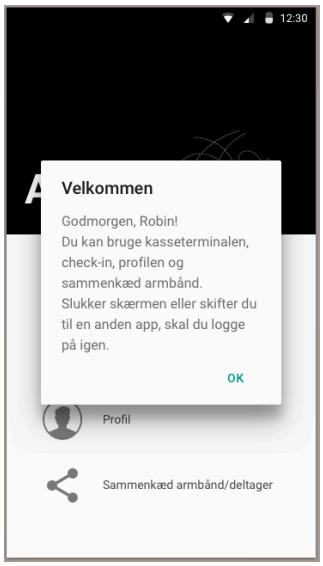


ALSSUND will decrypt the third value using ALS #42's public key and validate that the third value is an approved host and publish

`"HOST:APPROVED:A9BF2910CC0459AD:Robin:[1,2,3]"`

Should the approval fail ALSSUND will publish

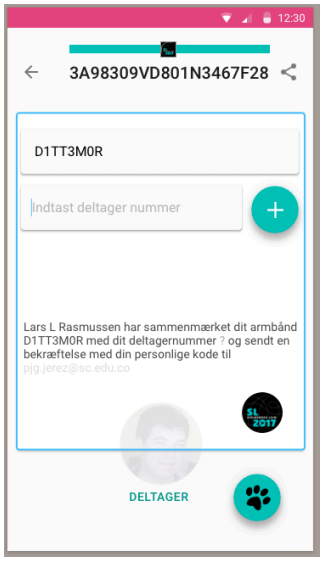
`"HOST:-1:REASON"`



The Wristband Linking Workflow

ALS will encrypt a wristband RFID, the 8 character kode, and the user number, and publish the message

`"WRISTBAND:NEW:A9BF2910CC0459AD"`



ALSSUND will decrypt the message using ALS #42 public key, verify the wristband RFID as not linked, link the RFID, the 8 character kode, and the user number to the wristband and return

`"WRISTBAND:LINKED:A9BF2910CC0459AD"`

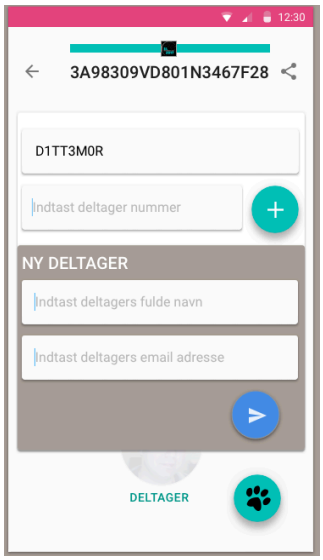
ALSSUND will send a confirmation email if the user has left a forwarding email address, with a password which the user will only need when building credit online.

In case the wristband does exist already, could not be inserted, or some other error surfaced, ALSSUND will respond with

`"WRISTBAND:-1:REASON"`

One such 'reason' could be: "USER UNKNOWN" in which case the host could opt to add the user by pressing the 'plus' sign next to the input field, and type in the user details and finish by pressing the 'paper plane' to send

ALS will publish



`"USER:NEW:1324908243F432E232CD0AF"`

ALSSUND will decrypt the third value to find a JSON object with a user name and an email address, and will publish

`"USER:53621:4E2FFA2"`

ALS will decrypt the third value using ALSSUND's public key and insert the number extracted from that operation into the input field

In case the user is not insertable (fg the email address already exist), ALSSUND will respond with

`"USER:-1:REASON"`

The Building Credit Workflow

ALS will encrypt the RFID and publish

```
“WRISTBAND:12397812368712398123098”
```

ALSSUND will decrypt the second value using the ALS #42 public key and lookup the ID part of the message and publish


```
“WRISTBAND:PABLO:\
Pablo Jesús Garcia Jeréz:/images/tmp/fer5tgaserga.png”
```

ALS will ask for transactions, publishing

```
“TRANSACTIONS:123978123687123981230 98”
```

ALSSUND will compile a JSON object and publish a tmp file reference

```
“TRANSACTIONS:/tmp/f3tymmewr2serga.enc”
```

This object looks like this

```
{ transactions: [
      transaction: {
            timestamp: “23/7/17 14:55”,
            what: “Slagter Bo”,
            amount: “22,75”
            host: “/images/hosts/portrait-424235-xs.png”
      },
      . . .,
      transaction: {},
] }
```

ALS will GET and decrypt the JSON object and display it. ALSSUND will delete the tmp file once a terminal has requested it - terminals will have to ask for transactions again, should transmissions be buggy!

Responding to the host adding funds, ALS will publish

```
“DEBITOR:123978123687123981230 98:CREDIT:200.00:EUR”
```

ALSSUND will build a new transaction and publish a message with the users new credit

```
“DEBITOR:123978123687123981230 98:CREDITED:1982.50:342093”
```

ALSSUND will send the bill as a snippet of HTML if the user has left a forwarding email address, with a password which the user will only need when building credit online.

If the transaction cannot be created (fx if the host and the user is one and the same) ALSSUND will publish

“DEBITOR:-1:REASON”

The Payment Transaction

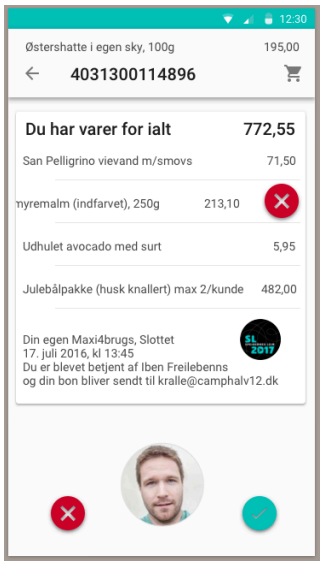
ALS will initiate the transaction by encrypting the RFID and publishing the message

`"WRISTBAND:12397812368712398123098"`

ALSSUND will decrypt the second value using the ALS #42 public key and lookup the ID part of the message and publish

`"WRISTBAND:ROBIN:/images/tmp/fer5tgaserga.png"`

after having pulled the user portrait and anonymised the filename.



ALS will publish

`"PRODUCT:9081237841232"`

ALSSUND will publish

`"PRODUCT:9081237841232:myremalm (indfarvet), 250g:25.75"`

Finally ALS will publish

`"BUYER:12397812368712398123098:CREDIT:148.95"`

ALSSUND will lock the user bank account, verify the credit limit, make a transaction, save the transaction id and publish

`"BUYER:12397812368712398123098:CREDITED:148.95:234523"`

and unlock the user bank account.

ALSSUND will send the bill as a snippet of HTML if the user has left a forwarding email address, with a password which the user will only need when building credit online.

If the transaction cannot be created (fx if the host and the user is one and the same) ALSSUND will publish

`"BUYER:-1:REASON"`