FoodCloud

TechUpdate

Overview

1. FoodCloud Intro
    a. Mission
    b. Some stats
2. FoodCloud Platform
    a. Version #1: Copia (launched Jan 2016)
    b. Version #2: Metro (schedule Q3 2018)
3. Tech Stack
    a. `Micro'services, bus-connected
    b. Scala, Play, Akka(-streams), RabbitMQ
4. The Guessing Game
    a. #NoEstimates
    b. Small increments of functionality
    c. Continuously tested and deployed

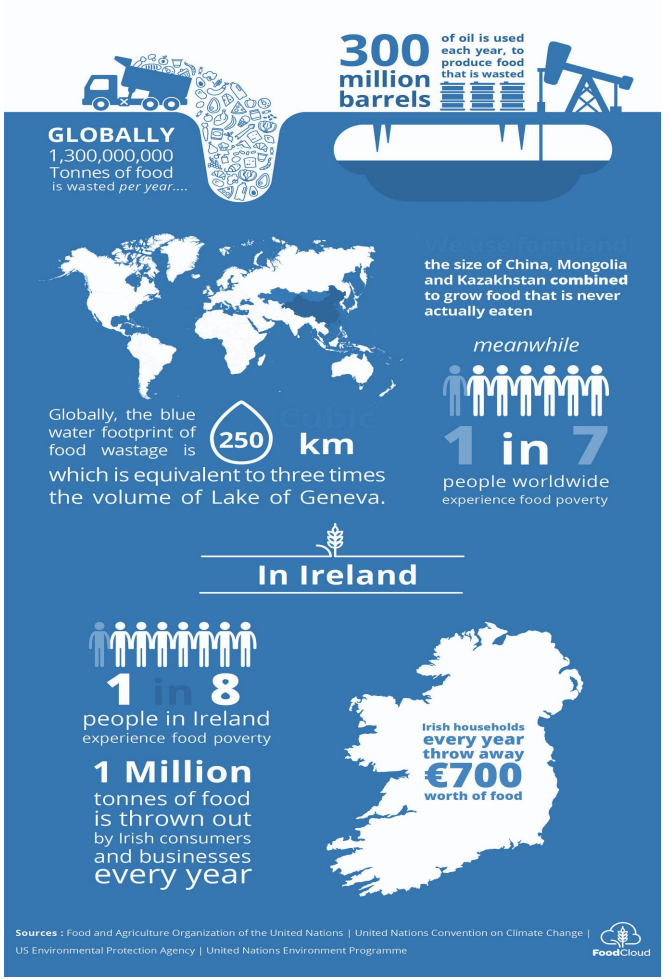
FoodCloud Intro

- Social Enterprise (not-for-profit)
- However, sustainable (revenue stream to support dev & ops)
- Started in Ireland, working with 18 Tesco stores, in 2013
- Expanded to UK, with Tesco, now c. 3,500 stores donating daily
- Work with Tesco, Aldi, Lidl, Dunnes, Waitrose (UK)
- Currently, weighted toward retailer needs (CSR, data)
- Now increasing support for Food banks, internationally
- Pilot projects in Eastern Europe and Australia
- Vision: many virtual food banks set-up by local ‘food heroes’ using our platform but with little or no support

Surplus Food Redistribution

FoodCloud contributes to solving food poverty by connecting businesses having surplus food, to charities in their local community. The solution uses technology to overcome some of the barriers that retailers have faced in donating perishable surplus food within communities.

FoodCloud, a not-for-profit social enterprise, funds its operations by charging retailers an annual fee for the service. Working closely with retailers to integrate with their in-store processes, whilst meeting requirements for donations, FoodCloud is able to deliver a cost-effective solution including forward-traceability.



Impact to date

KG

12,961,216

KG of food saved
in Ireland and the UK

28,514,675

meals redistributed
to charitable causes

7,500+

Charity and Community
groups work with us

4,000+

Food and Retail
partners work with us

The Process

Retailer has surplus

Charity gets notification

FoodCloud

Charity collects

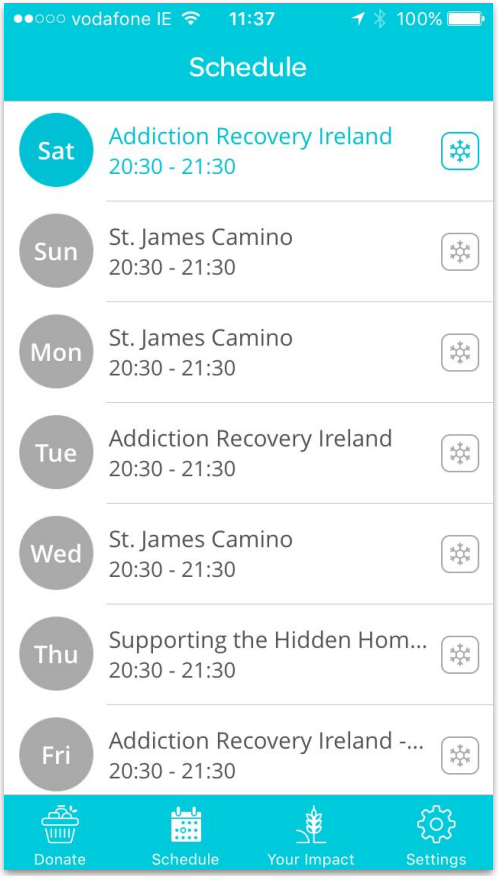
Outcome: waste eliminated, people fed

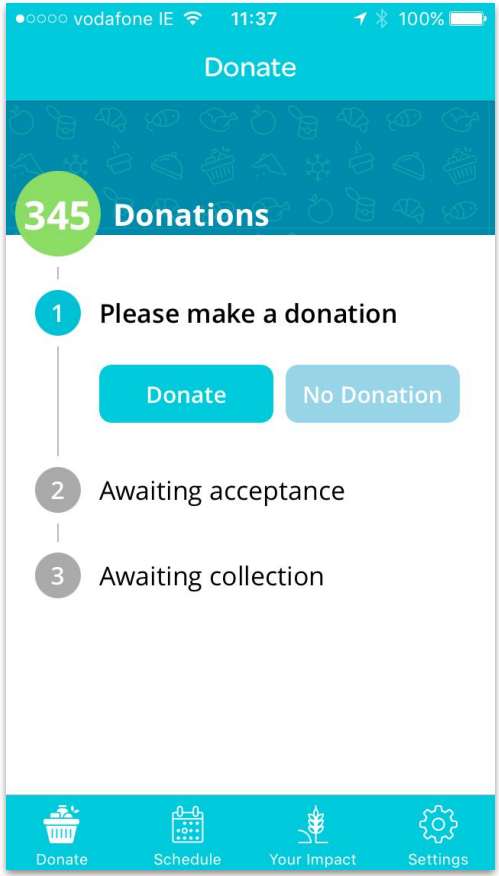
FoodCloud Platform

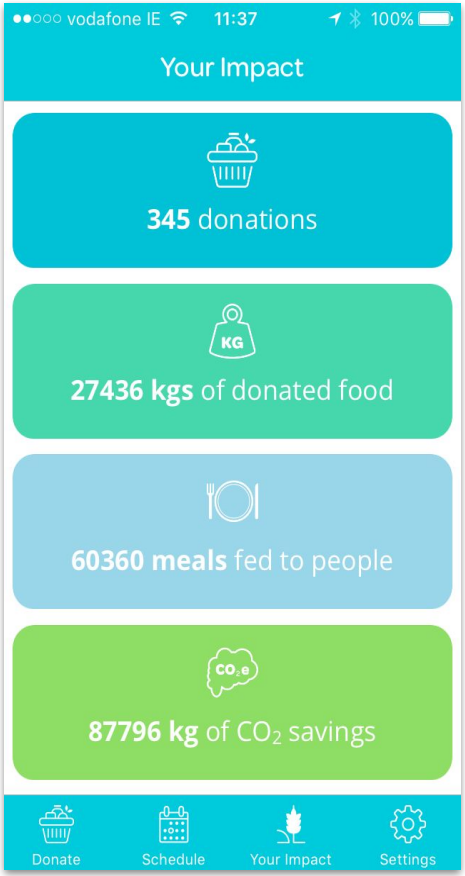
- Retailer integrations & communications
- Partner management
- Donation scheduling & management
- Support dashboard
- ‘Fairness’ algorithm
- Charity communications

Availability
Web & Mobile access
Internationalization

FoodCloud Donor app

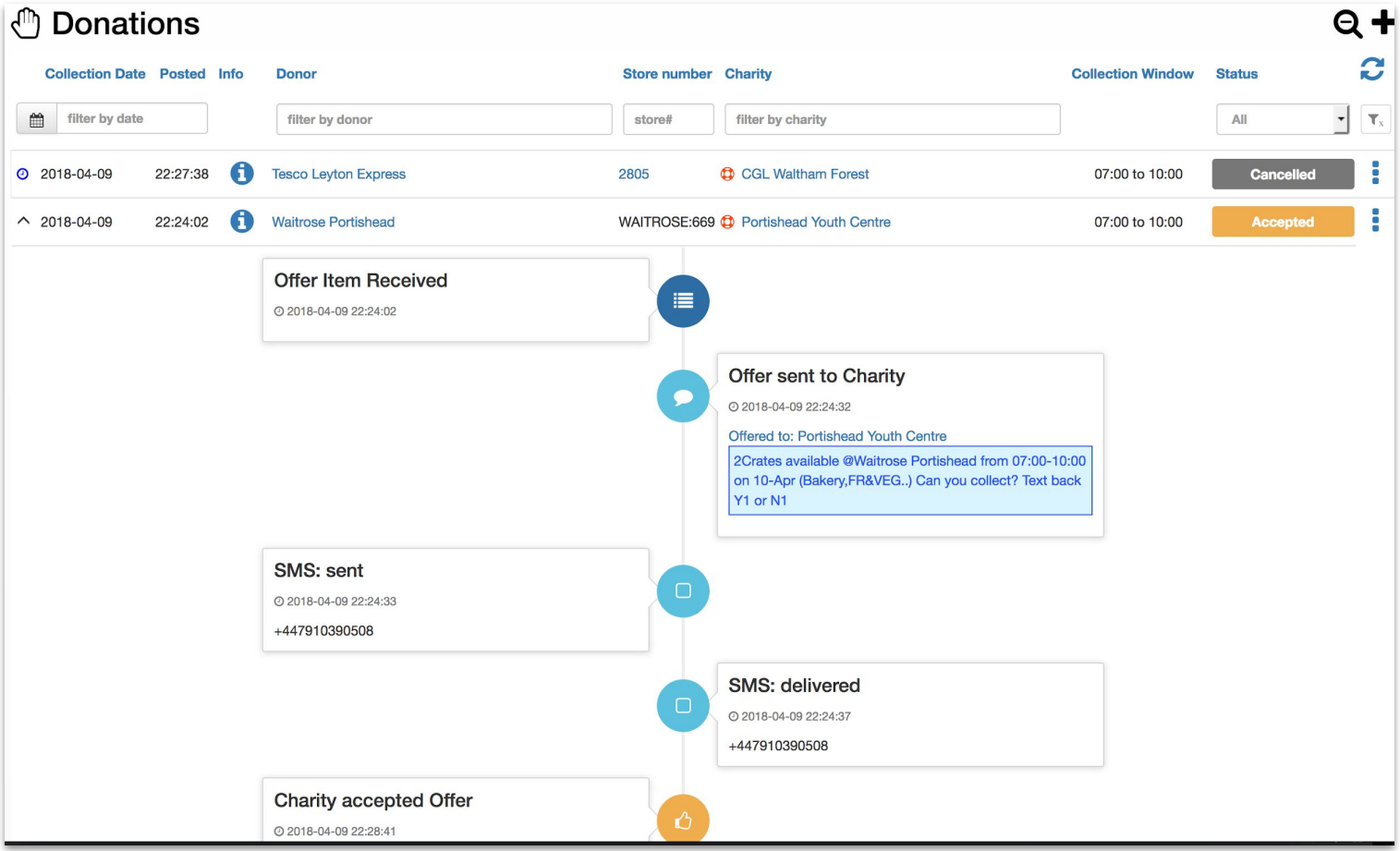






FoodCloud Charity SMS/app

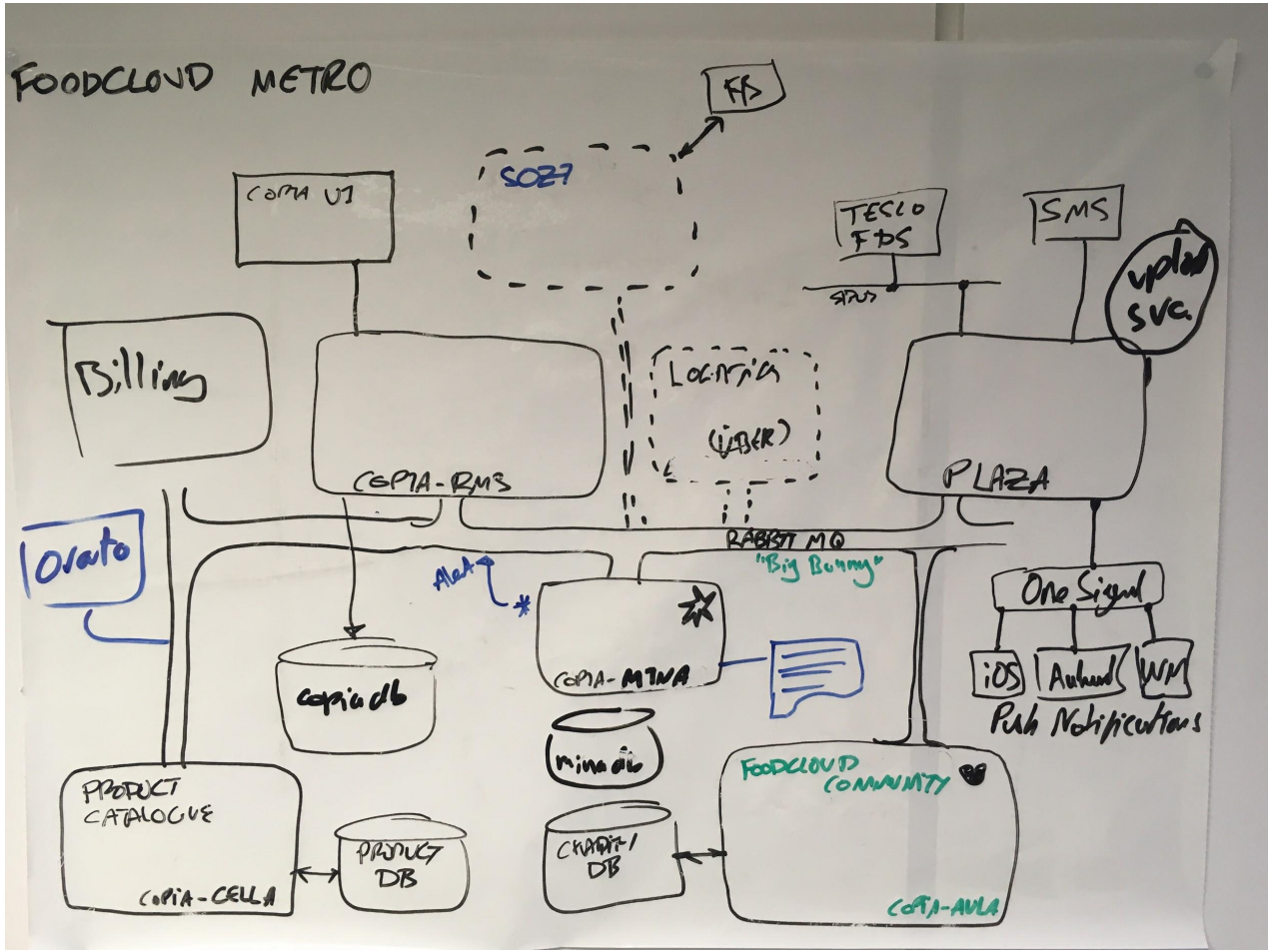
FoodCloud Support



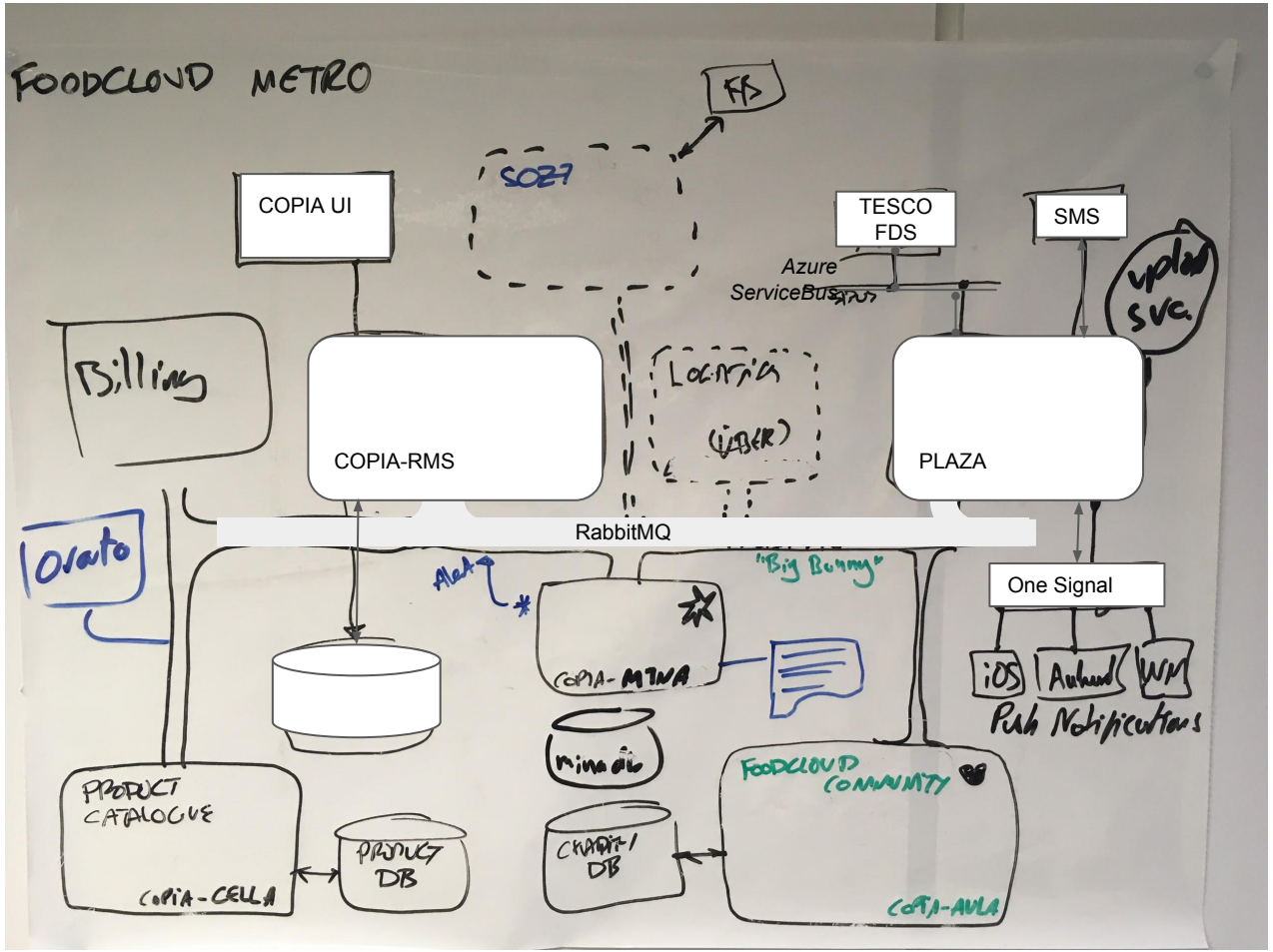
Platform Evolution: SRP

- Started with a minimal two-module solution (‘duolithic’?)
  - Copia: donation processing, support & partner management
  - Plaza: integration gateway
- Analytics became important, added:
  - Mina: event store module
- Integrated phone support added
- Retailer & Charity management becoming more involved
  - Aula (Community): Flexible metadata based organization DB
  - Single-sign on directory
- Scaling both the web app and background processing a challenge
  - Extract ‘donation engine’, re-write to be more reactive
  - Re-build web front-end with React.JS (leverage web sockets)

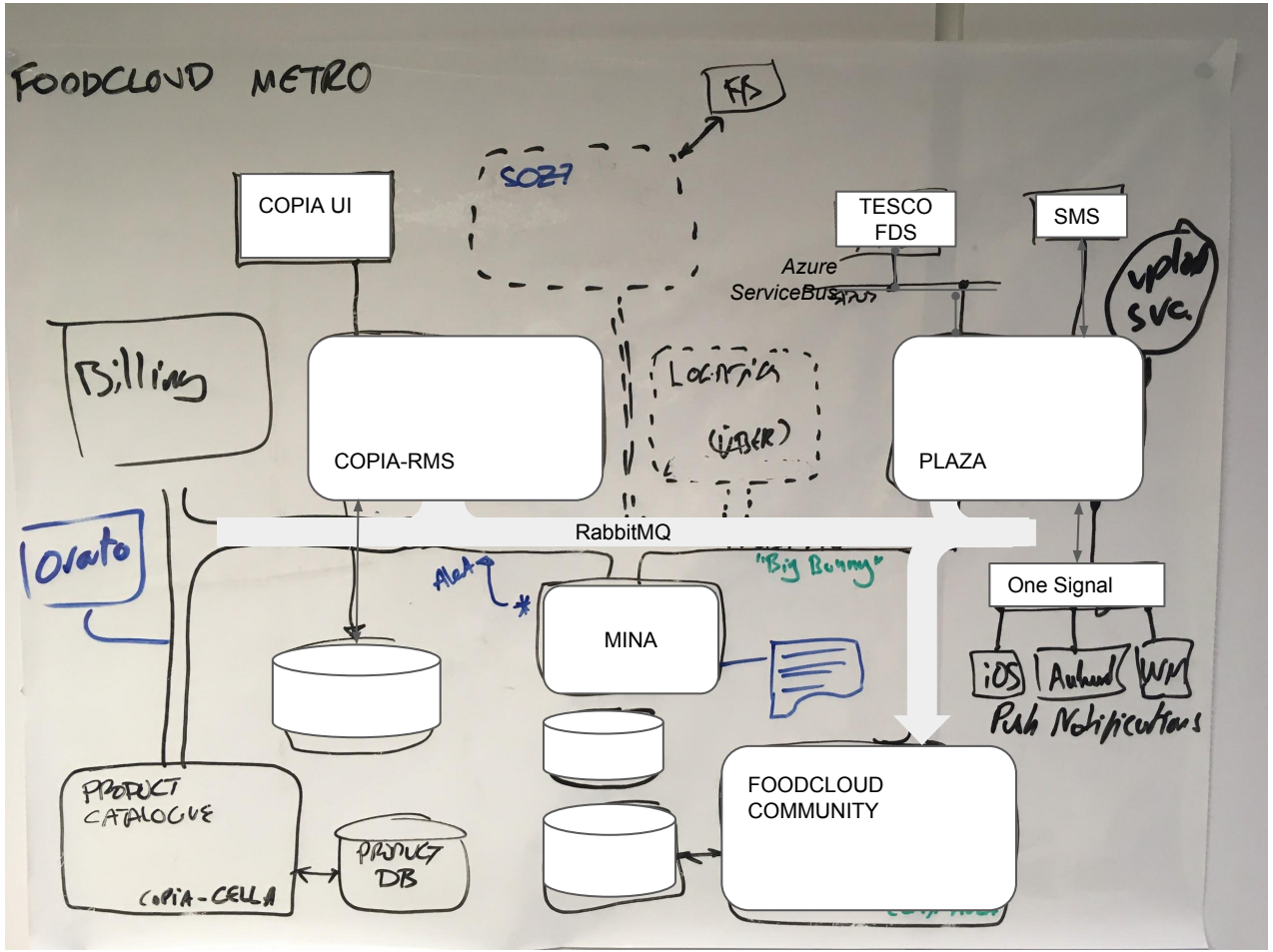
District Planning



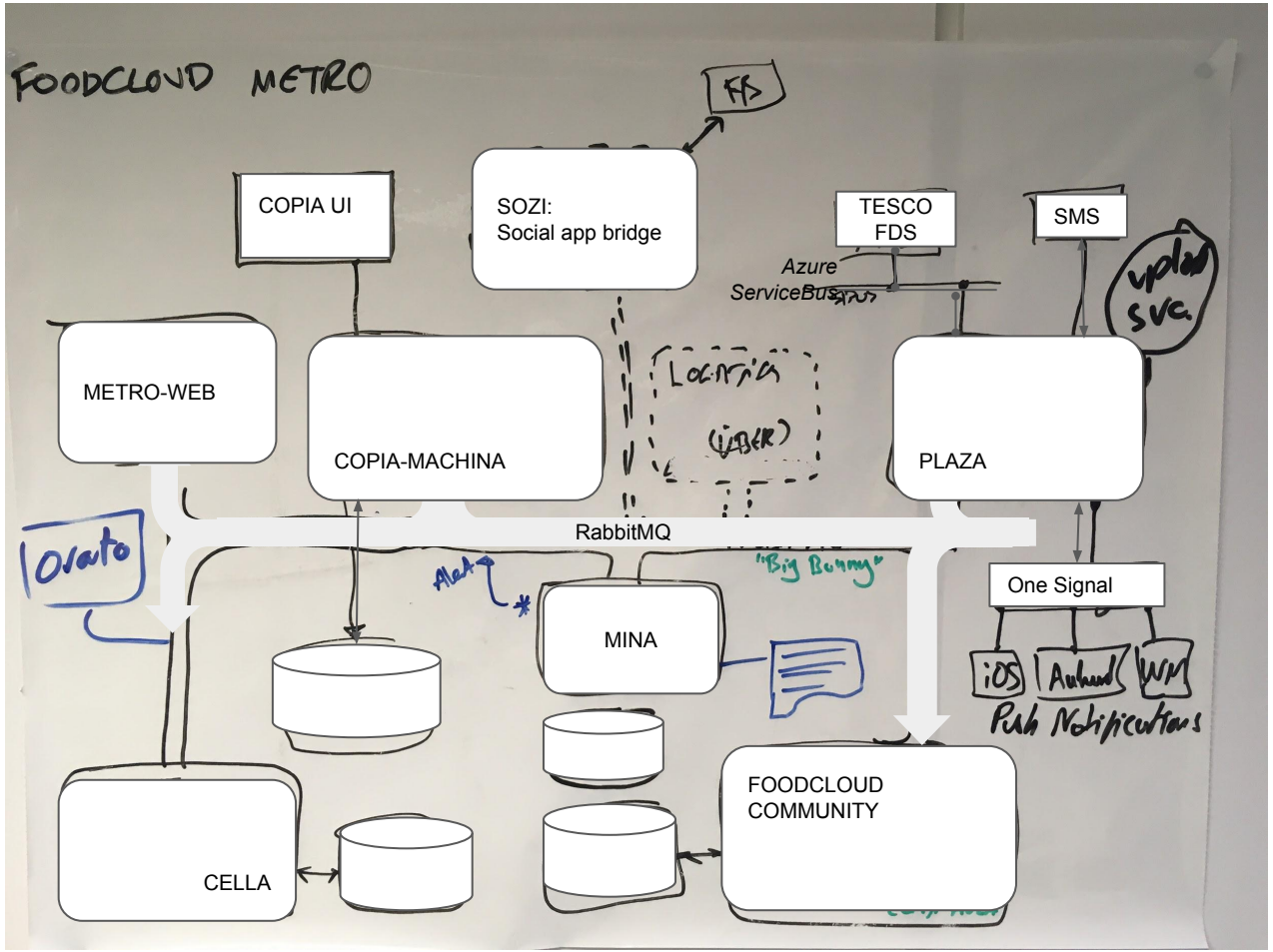
District Planning: iteration #1

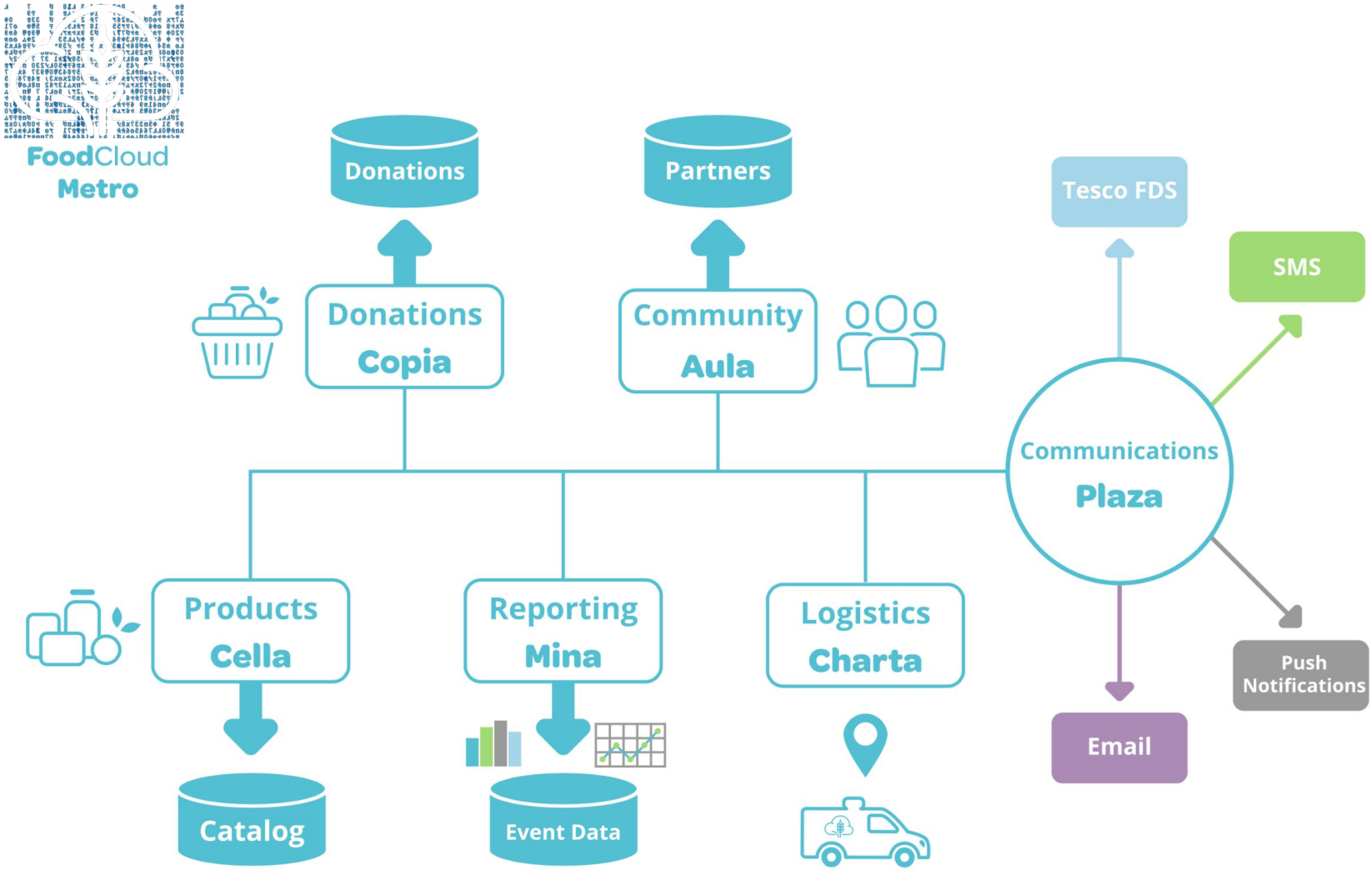


District Planning: iteration #2



District Planning: iteration #3




FoodCloud
Metro
Donations
Partners
Tesco FDS
SMS
Donations
Copia
Community
Aula
Communications
Plaza
Products
Cella
Reporting
Mina
Logistics
Charta
Push
Notifications
Email
Catalog
Event Data

Tech Stack

- Supports:
  - Rapid innovation
  - A small team (and low costs)
  - Customer/user feedback

- Scala
- Akka-streams
- Play framework
- Slick FRM + PostgreSql
- Angular ~> React.JS
- Heroku PaaS
- Android/iOS

Bonus Donations

```
def autoOfferBonus(donation: Donation, donor: DonorView): Future[Option[Charity]] = {
 require(donor.offersBonus, s"donor should have Bonus Donation flag set in $donor")

 val source = Source.fromPublisher(charityService.publishFairMatch(donor.orgId))

 val findCharity =
    Flow[CharityFairnessView].mapAsync(1) ( cfv =>
      charityService.find(cfv.orgId) map ( (cfv, )))
   .collect { case (cfv,Some(charity)) => (cfv,charity)}

 val findOffers = Flow[(CharityFairnessView,Charity)].mapAsync(1) { case (cfv, charity) =>
   offerService.findByCharityAndDonation(cfv.orgId, donation.id) map ((cfv, charity, _))
 }

 val suitability = Flow[(CharityFairnessView,Charity,Seq[Offer])].filter {
   case (cfv,charity,offers) =>
     !cfv.isClosed && cfv.collects && charity.acceptsBonus && offers.isEmpty
 }.map(_._2)

 source via findCharity via findOffers via suitability runWith Sink.headOption
}
```

The Guessing game

"Estimation is very difficult, perhaps impossible, and often misused" - Ron Jeffries

- #NoEstimates
  - "in a complex environment, we don't have discernible causality." (*Vasco Duarte)*
  - predict the future in a complex and changing environment?
  - often padded by developers, encouraging mistrust
  - waste of valuable time
- Small increments of *holistic* functionality
- Continuously tested and deployed
- Write some code ~ it doesn't take long

Some hints

- Rather than providing an estimate about how much time it will take, dive into the work by identifying the most important story and implementing it. Once that's completed, move on to the next most important story.
- Work in small chunks, you often negate the need for many items in the requirements document. Since many of the items the client initially wants end up being set aside, providing estimates on them is a waste of time. (Why estimate when chances are good you won't end up completing the work at all?)
- Move beyond the idea of story points and, instead, look at the total number of stories that need to be completed as a gauge of the total amount of work that needs to be done.

Woody Zuill