Brief for ‘beans’

About the project

Eating well and living mindfully leads away from supermarkets with their long supply chains, squeezing of farmers and chasing of shareholder profits toward small groups of people cooperating to buy from trusted farmers in their area. To source organic food locally and reliably though is quite complex especially on the scale of 10 - 30 members of a typical food group. What is needed is an app which takes away much of the pain of organising custom orders of regular local ethical food.

A sophisticated and intuitive app which enables the many facets of a food group to work smoothly could also then be a major help in others wishing to adopt the local food group approach. This is not a lucrative business opportunity: there is not more than $50 per week per group for this kind of app. However for those who want the freshest most nutritious food grown in a way that builds soil and gives local farmers a fair return this app would soon become essential.

Russell & Gabrielle Austerberry have been involved in such a group for 5 years, using the online web app Lettuceshare. While ensuring $1000 - $2000 of custom orders per fortnight within a group of 30 people would be impossible without it, there are many places where friction could be reduced. While ‘beans’ will be developed independently, there remains a cordial relationship between Russell and Mal Blaney (author of Lettuceshare) and the possibility of collaborating down the track.

Project goals

1. An easy-to-administer management tool for food group organisers
2. An easy-to-use ordering and reminder tool for food group members
3. A purchaseable app that can easily be set up for each new group who is interested

About the team

Russell Austerberry is ‘the team’. He has extensive knowledge of administration of food groups (>5 years) as well as the major part of a year skilling up as web app developer in preparation for developing ‘beans’.

In Scope

1. A fast, on-the-go, good-looking easy to use app that works well even with flaky internet.
2. An app that assists admin and members at every stage of the order cycle:
    a. Availability (importing supplier availability)

   b. Ordering (intuitive shopping cart experience)
   c. Placing Orders to Suppliers (generate orders, track over/under/not available)
   d. Delivery (supplier printouts)
   e. Packing (management of teams, reminders, packing lists, adjustments)
   f. Invoicing (sales & supplier invoices, payments, financial tracking)
3. A flexible array of real-time reporting features for both members (eg balances) and admins (eg sales figures)

Not in Scope

1. Integrating app with accounting package or banking software
2. Stock tracking and other features which may make this app suitable for wholesalers (not yet)

Timeline

Given this is a 'love' project the timeline is as long as a piece of string. It will be progressed as and when possible along from milestone to milestone (see Plan section of document).

Approach

- Further requirements analysis will be conducted within each milestone as listed in Plan section, and changes or additions made as appropriate.
- A component/milestone is "done" once
  - Component is feature complete and tested working by Russell
  - feedback has been received by three independent users
  - subsequent bug fixes or features have been implemented and re-tested.
  - Code is well commented and refactored to be as elegant as possible
  - All relevant changes have been recorded in this documentation
  - Stable version has been saved & backed up
- Whole project is to be tracked in TeamGantt app
- Each component/milestone to be planned/tracked using Trello
- User stories added and mapped using Board Thing

Plan

Feature summary

1. A responsive, intuitive, speedy offline-first, mobile-first web app for managing a local food group
2. Admin features:
    a. Calendar based order cycle management
    b. Flexible import and translation of various supplier lists to standard format ‘beans’ taxonomy
    c. Flexible and fast multi-level sorting, culling, modifying and replacement of items
    d. Multifaceted, modifiable and flexible item pricing structure
    e. Management of packing teams and other roles
    f. Print-ready exports of various lists including delivered items, member packing slips, and supplier checklists
    g. Reports suitable for accounting & audit at any time
    h. Auto-invoicing, member balance tracking and payment reminders
    i. Easy reconciliation with group bank account
3. Member features:
    a. Easy to use custom online ordering
    b. ‘Demo mode’ when outside ordering window
    c. In app notifications for upcoming ordering, packing or other events
    d. Simple customisable recurring orders
    e. Accessible invoice, payment and balance history on demand

Milestones

See Teamweek gantt chart for planning & progress. The following is high level overview:

1. Preliminaries:
    a. Brief
        i. Goals
        ii. Scope
        iii. Roadmap
    b. Plan
        i. Feature summary
        ii. User stories & flow
        iii. Wireframes
        iv. Tech stack
        v. Database model

2. Availability component
    a. Development environment, version control
    b. Core database implementation
    c. Initial pages with ‘look and feel’
    d. Import supplier availability lists & parse to ‘beans’ taxonomy
3. Admin functionality component
    a. Order cycle management & setup wizard
    b. Main admin interface
4. Ordering & accounts component
    a. Member login & security
    b. Shopping cart
    c. Invoicing
    d. Finances
5. Notifications & reports component
    a. Formatted packing & item slips
    b. Finance & other reports
    c. Notification & announcement system
6. Deployment
    a. Alpha testing
    b. Monetisation functionality
    c. Multiple rollout functionality
7. Maintenance
    a. Beta testing
    b. Rollout to other groups
    c. Features added or bugs fixed as needed

Users

1. Member
    a. Packing
    b. Ordering
2. Internal Roles
    a. Organiser
    b. Making available
    c. Promotion
    d. Finance
    e. Ordering
    f. Pickup
    g. Adjustment
    h. Steering committee
3. Supplier
4. Auditor
5. Compliance entity

User stories

Business logic

Setup

1. Organiser enters or adopts a taxonomy (populated list of items each having category, type and variety)

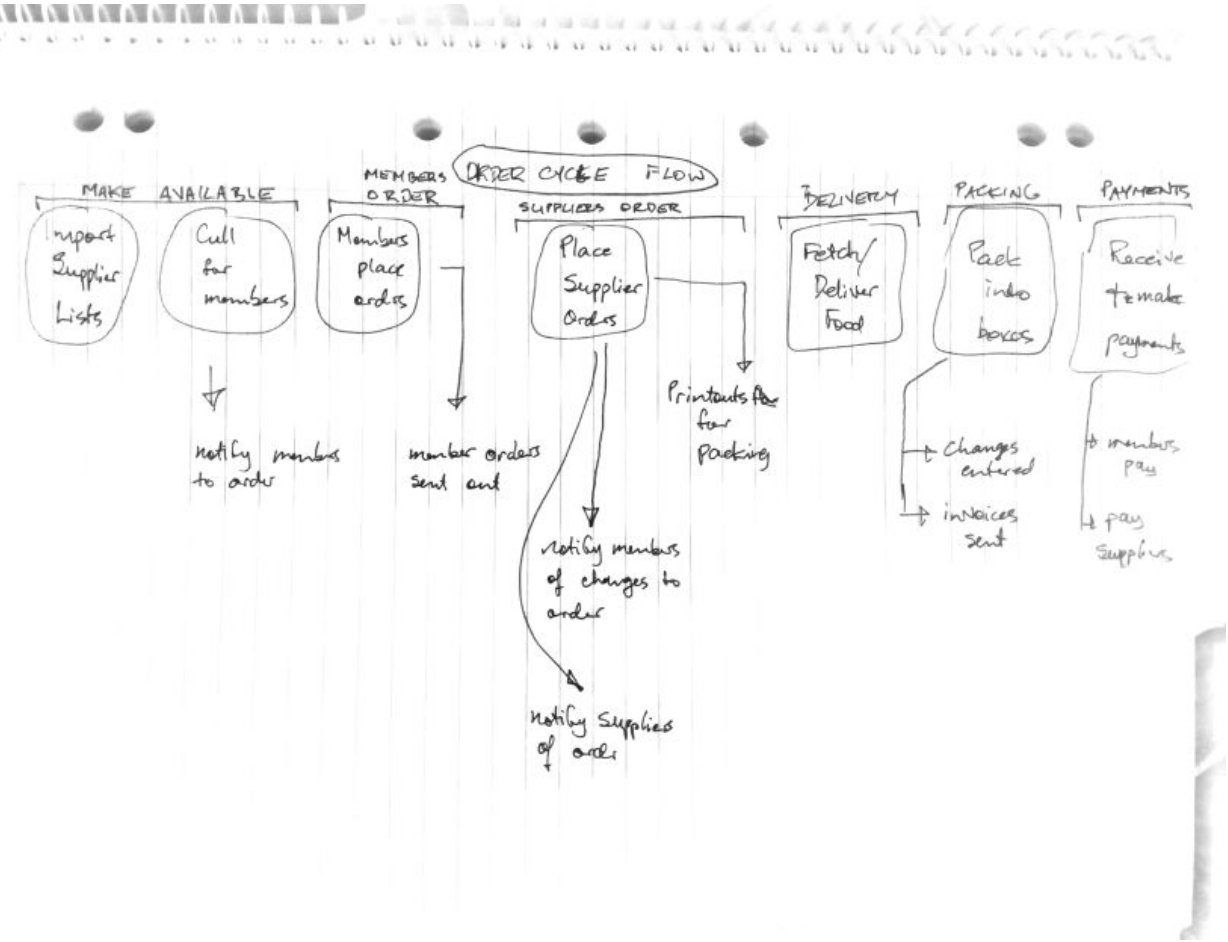
Order cycle

1. For each suppliler, either manual update OR
    a. Availability setter imports supplier list
    b. App matches or adds (with availability setter approval) each available item to one in the taxonomy
    c. App (with approval from availability setter) parses items and other details (price, unit, boxSize etc) into supplier availability list
2. Availability setter culls available items
3. Availability setter enters supplier and producer notes, any other news
4. App notifies members of order window open + news
5. Members' online orders captured
6. From order summary data, orderer places orders with suppliers (via app)
7. Orderer records totals ordered (if different)
8. Orderer updates order list items (NA, swaps)
9. App calculates extras & NA, prompts orderer for news
10. App notifies members of
    a. Their order, with changes from original
    b. News
    c. Duties on packing team
    d. Member requests (eg swap packing duties)
11. Adjuster enters late orders of extras only
12. App prepares printable items for packing event:
    a. Member orders
    b. Packing list
    c. Extras sheet
    d. Supplier order checklist
13. Total order fetched or delivered
    a. Changes or NA items clearly marked on paper
14. Team packs into boxes, recording changes on paper
15. Members pick up food, return empty boxes
16. Adjuster enters all changes to orders

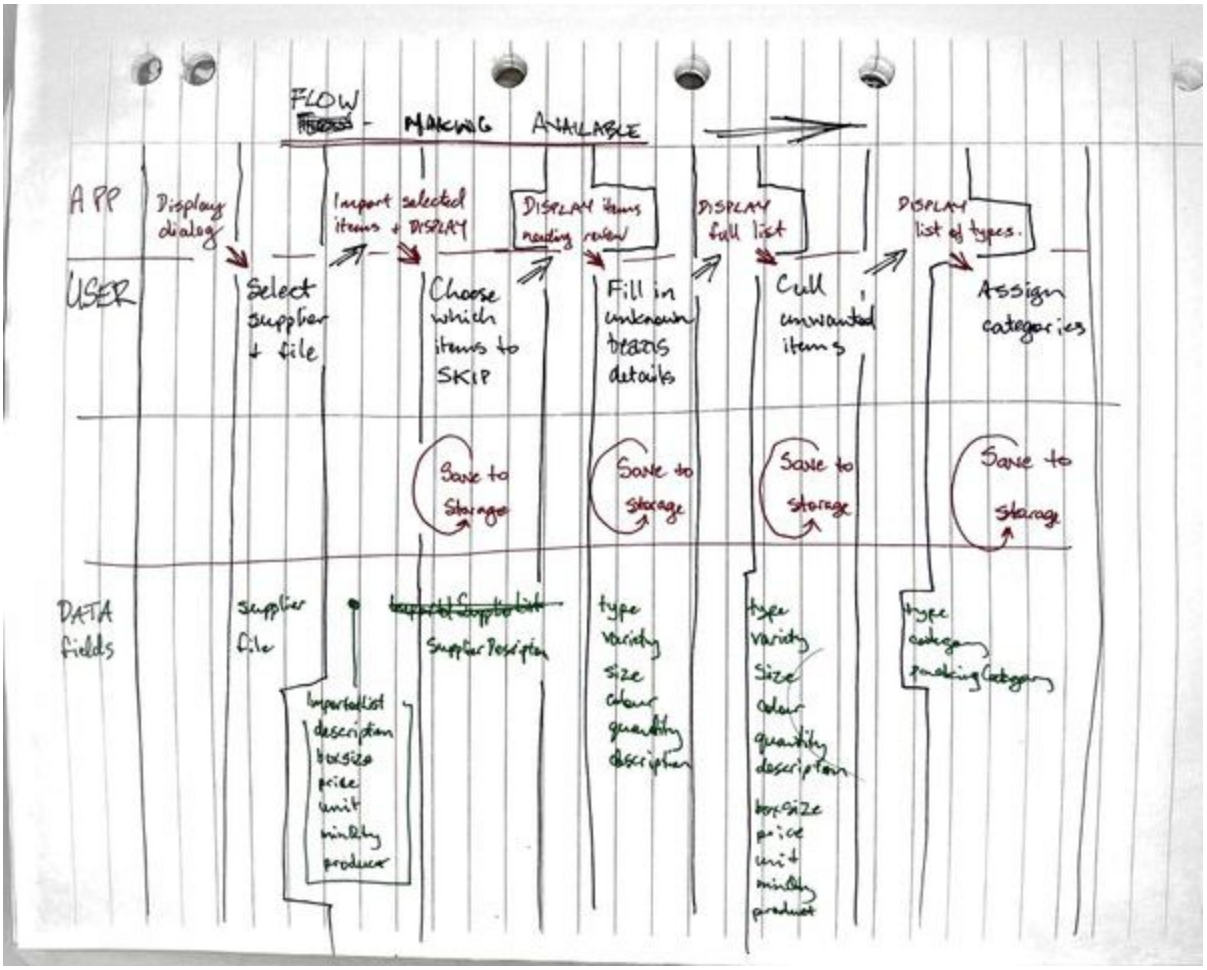
17. Adjuster records supplier invoices or other receipts in app (pic or email)
18. Invoices sent out:
    a. App calculates and sends invoices to members
    b. PDF copy of invoice saved
    c. Summary data for invoice saved (total amount, balance)
    d. Adjuster fields outstanding queries, modifies orders
        i. Upon modification, new invoice sent to member & saved in app
19. Members pay into bank account
20. Financier records payments
    a. App updates
        i. invoice balances
        ii. Member balances
21. Order cycle closed off
    a. Summary data recorded:
        i. Total sales
        ii. Total other income
        iii. Total members ordered
        iv. Total suppliers paid
        v. Total other expenses
        vi. Total expenses paid
        vii. Total members owing
        viii. Total members credit
        ix. Total active members
        x. Total assets
        xi. Total liabilities
    b. Member ‘live’ order data erased (order PDF still on record)
    c. App notifies financier with report summary

Concepts

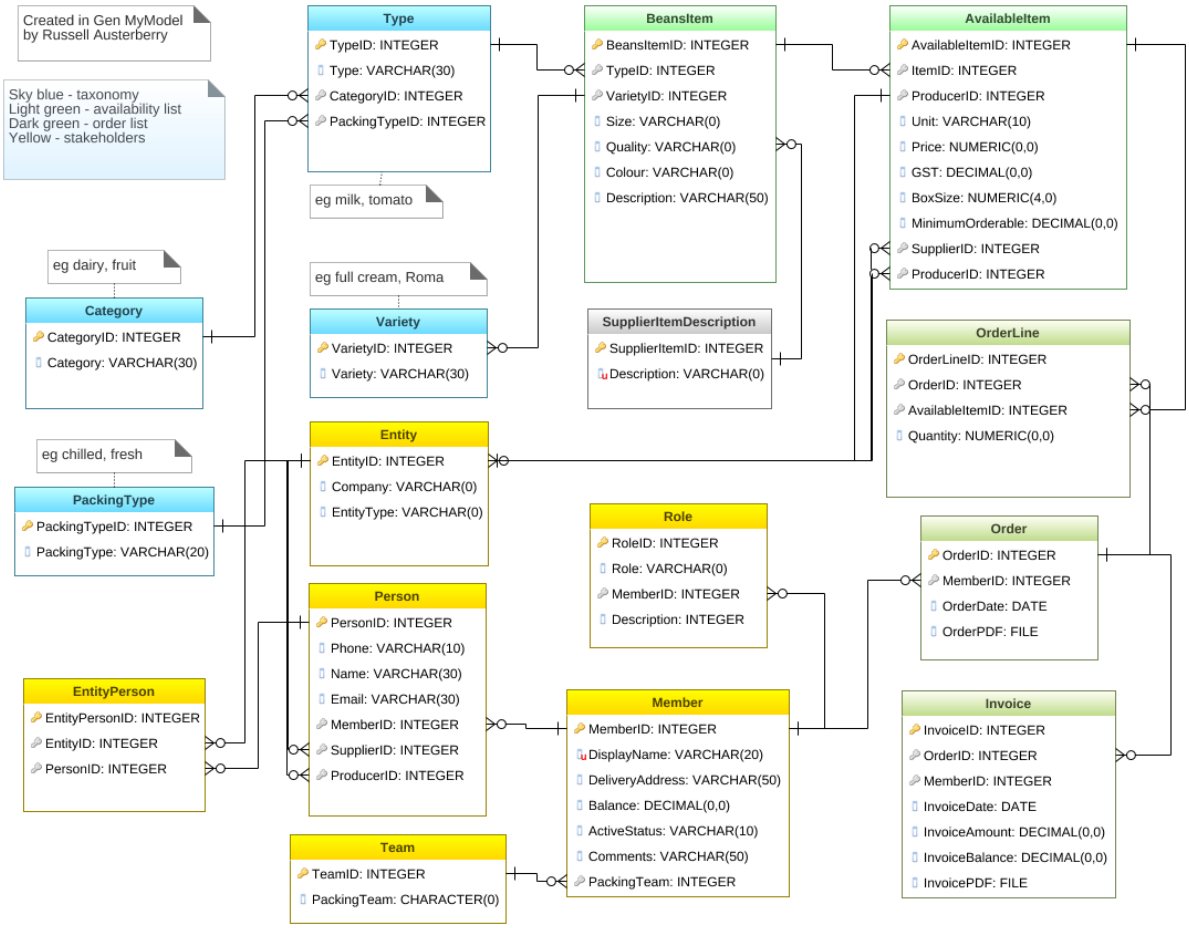
Overview of order cycle (high level)



Overview - making available. Interaction between user and app, with data fields listed



Data model



Tech stack

Frontend

- HTML, CSS, JavaScript
- Relational-pouch
- PouchDB

Backend

- Node.js
- Express
- CouchDB

Development environment

- Ubuntu 19 box
- Visual Studio Code IDE
- Git

Production environment

To be determined

Architecture

Looking for client-side relational database that can also sync to server.
Sync to server is ESSENTIAL, relational is (very) NICE TO HAVE

Best fit looks to be :
App.js <-> Relational-pouch <-> PouchDB <-| sync |-> CouchDB

With possible 2nd choice being AlaSQL instead of relational-pouch
This will allow

- *offline first* experience (on-device storage and querying for speed and reliability even if internet is down)
- *Server interaction* (for initial download, updating available products etc, syncing)

~~Start here. Intrigued by IndexedDB with relational Lovefield library~~
~~Dexie - wrapper for IndexedDB for more concise less error prone code. No sync?~~

~~Hoodie might also be a good fit, although possibly too early in development~~
Pouchdb is JavaScript emulation of CouchDB and strong in syncing (which sounds good…)
PouchDB runs client-side but is object based.. However you can use Relational Pouch with it…

- Uses IndexdDB and falls back to WebSQL as needed
- If neither of these supported, fall back to live CouchDB (server side db)
- However… relational-pouch doesn't have much action on github
- Thing is… PouchDB *syncs*. Sync may in the end be even more important than being relational.

AlaSQL - AlaSQL applies SQL operations to JavaScript arrays and objects therefore the library can best be described as a JavaScript SQL database.

- JavaScript data manipulation plus advanced filtering, grouping and joining
- Client-side SQL database with option for persistency (as Local Storage and Indexed DB)
- Plays nice with Excel and Google spreadsheets (and presumably .csv)
- Export, store, and import data from localStorage, IndexedDB, or Excel.
- Fast in-memory processing

- Still doesn't address *syncing* to server
- … can we play with AlaSQL as 'api' for PouchDB (which wraps IndexedDB)?
- … better off using the purpose built relational-pouch
- … consider AlaSQL if relational-pouch is inadequate

~~Lovefield - simple relational database for javascript … but no commits since 2017~~
~~Taffy looks simple but not necessarily relational~~

~~Sequelize ORM (Object Relational Mapper) for Postgres, MySQL, MariaDB, SQLite and Microsoft SQL Server. Warnings abound… Let's leave ORM alone for now~~
~~TypeORM. Fresh crop of ORM~~

~~Good ole' SQL~~

~~Parse can handle relational data…~~
~~Knex SQL query builder for JavaScript~~