DDD
a Todo app

My Journey
at ZinZen.me

Me (or any developer) at some point:

- Problem:
  - I need a Todo app…
    … but nothing fits my needs.
- Solution:
  - I'll make one myself!
    It's not that hard – and I already have a name.

Current status

- Still tinkering…

… many versions later ☺

initial commit

TL committed 6 years ago

1. *That* bad?
2. *That* hard?
3. *So little* spare time?
4. Crazy?

You *may* judge.
But first,
let's go way back…

Domain model - version 1 - 2016

```rust
struct Todo {
    id: Uuid,
    title: String,
    completed: bool,
}

impl Repository {
    pub fn add(todo: Todo) {}
    pub fn remove(todo: Todo) {}
    pub fn complete(todo: Todo) {}
}
```

This is an anemic
or CRUD domain model.

Perfect for
simple domains

- when people already
think this way.

Most people stop here…

- What can I say…

I’m stubborn?

Well…

<u>most</u> people

don’t

think CRUD …

```
todo_repository.add(‘get TOTK’);
```

So how do people *really* think?

I want to realize Dreams,
by achieving Goals,
scheduled as Tasks.

This is a slightly more complicated…
… but very doable!

Domain model - version 2 - 2017

Let’s cheat a bit … and say Dreams are just fuzzy Goals -

… then we only need two-level hierarchy:

$$\begin{tikzcd} \text{Goal} \arrow[d, shift right] \arrow[d, shift left] \\ \text{Tasks} \end{tikzcd}$$

```
struct Task {
    goal_id: Uuid,
    start: DateTime,
    end: DateTime,
}
```

I schedule Tasks
by selecting a Day
… and optionally a Time.
(assume 0:00 == no Time)

Right? It’s how ‘pro’ Todo apps work…

There – I'm done!

People are paying for this…

…so,
it *must* be good?

…of course, once I started thinking…

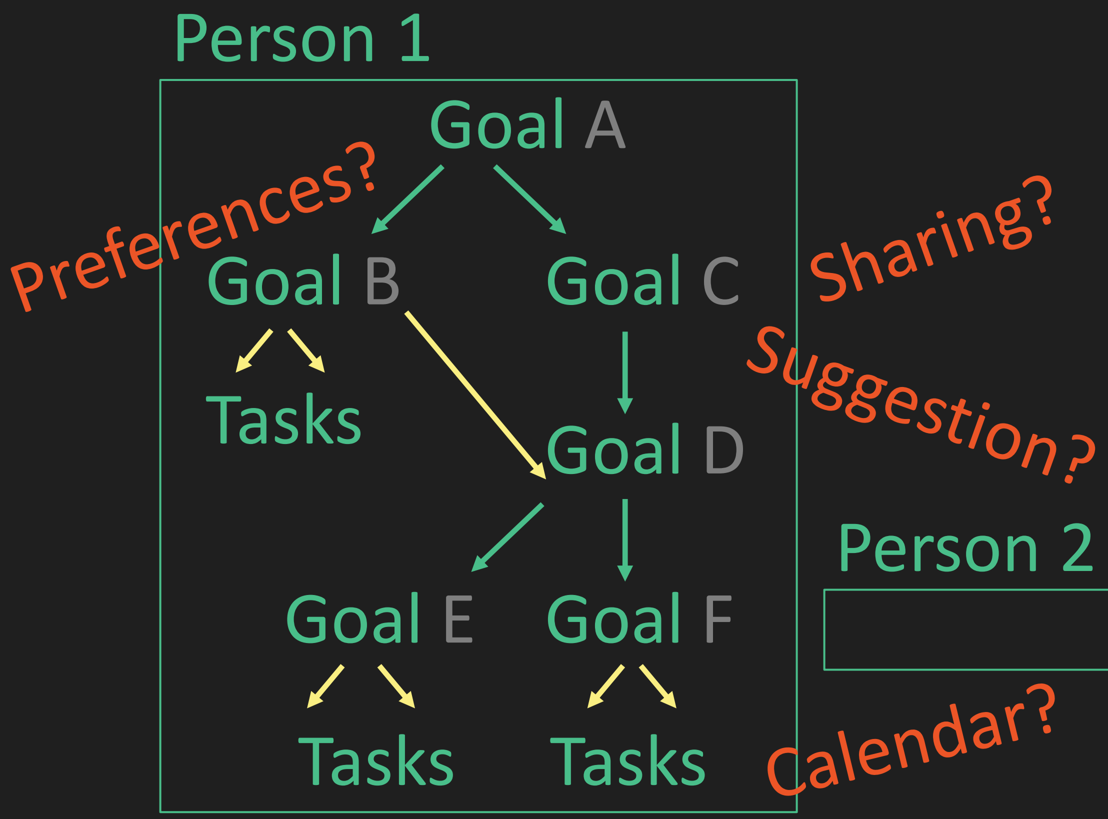
Fuzzy Goals like ‘make a living’ can have nested sub-Goals
with scheduling Preferences
- like ‘Project A on weekdays, daytime’
- or even be flexible - like ‘0-9 hours a day, 40h per week’

because I don’t want to manually (re)schedule stuff (over and over).

I also want to organize my Goals in a graph.

I also want to collaborate with Others
by sharing Goals/Tasks and accepting Suggestions
to improve my Goals and optimize my Calendar.

My mind, seconds later:



To put it simply …

I needed Donna.

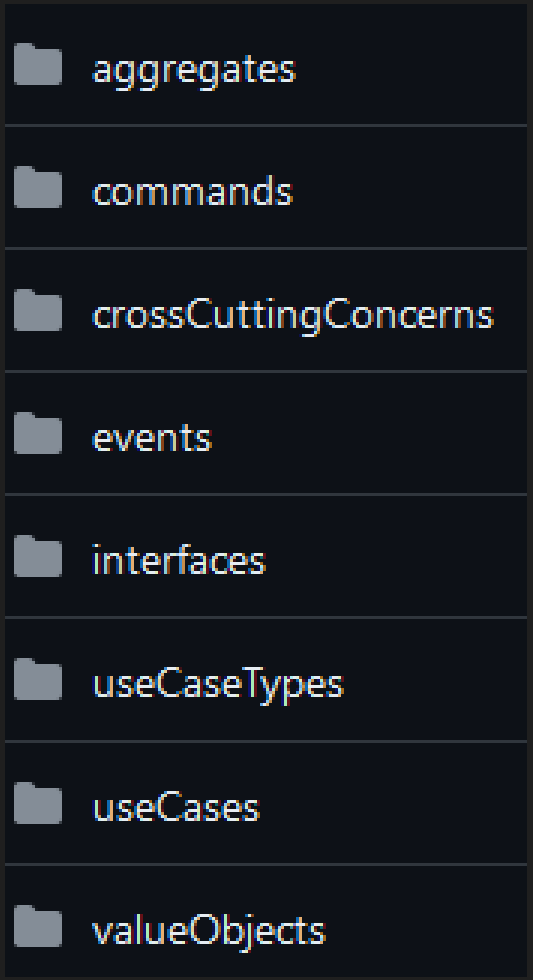
Sorry.

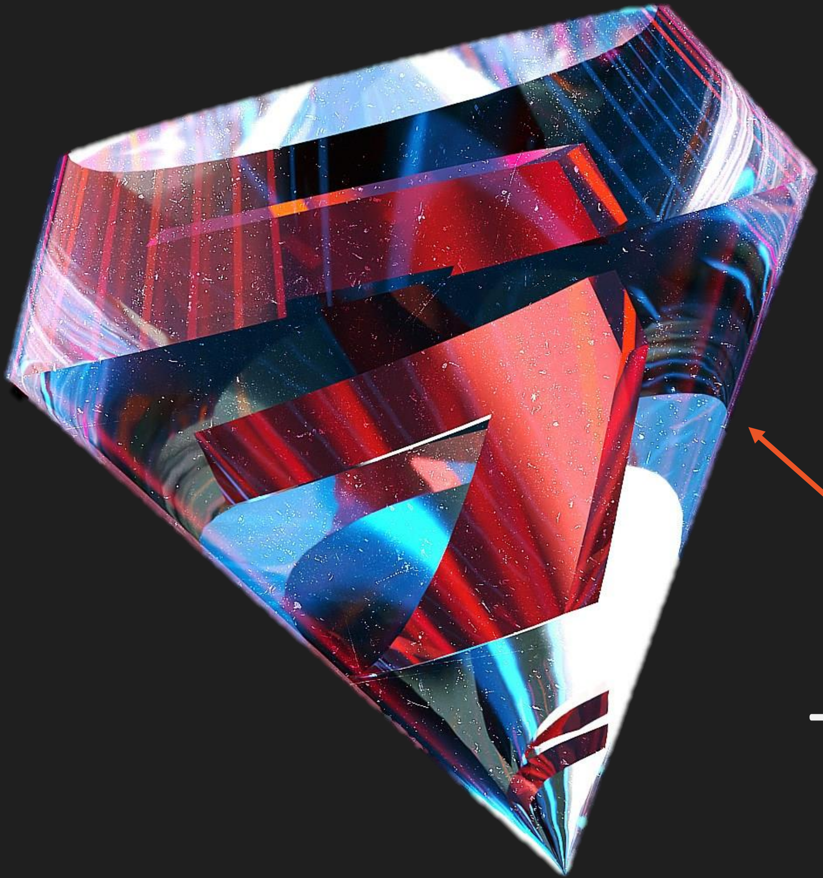
What’s the other option? Oh yeah, the app.

OK. Now it's complicated.

Let's try DDD!

So, I skipped DDD and started coding - 2018

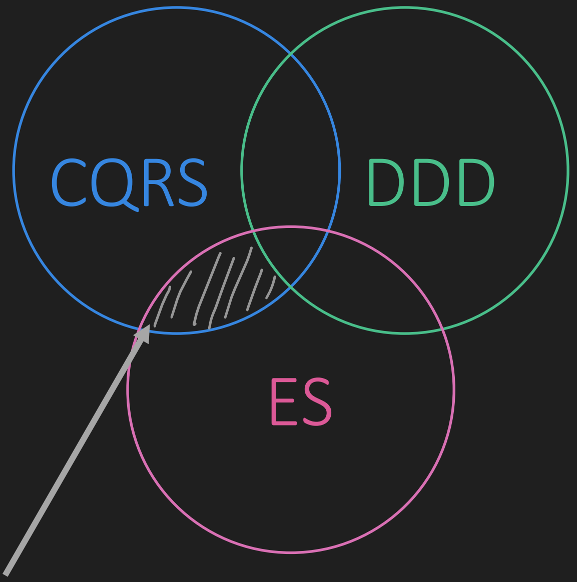

aggregates
commands
crossCuttingConcerns
events
interfaces
useCaseTypes
useCases
valueObjects



Shiny CQRS
+ Event sourcing pattern

… yes, I CQRS'd + Event Sourced everything

- This can be combined with DDD, but is not the same as doing DDD
  It was an interesting experience… and I learned a lot.



I was lost here.

Intermezzo - 2020

“No comment.”

...and still lost.

By 2021 - I had tried a few tech stacks

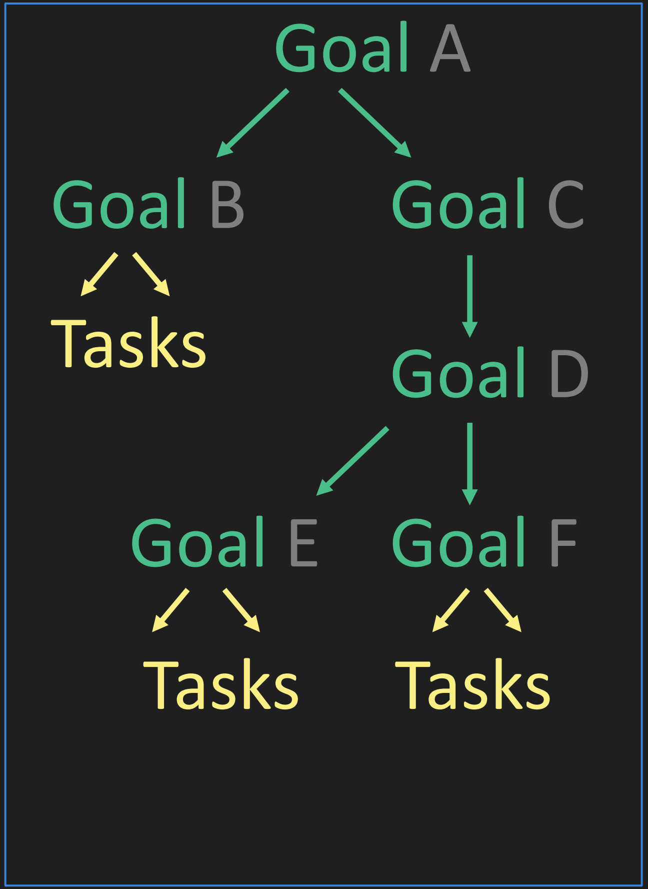
- A QT / C++ app
  - Snappy! License issue.
- A CQRS/Event-sourced Android app
  - Interesting. Slow. Boilerplate.
- A cloud-based central graph database with reactive vanilla JS frontend
  - … interesting opaque cloud challenges. Get support. ☺ Slow. Complex. Expensive.

No single stack covered all needs.

What if we aligned our code paradigm with domain properties?

The first rule of distributed computing...

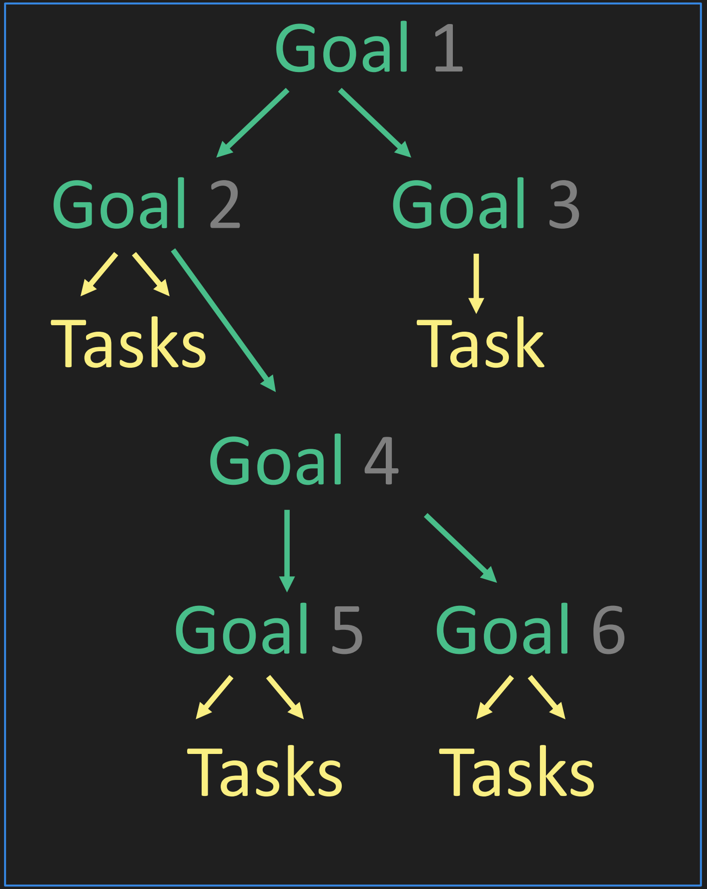
Alice’s device



Don’t fight
‘natural boundaries’

an offline-first PWA
person’s device domain
+
fast functional WASM
scheduler domain
+
↔ ‘dumb’ message pipe ↔
collaboration domain

Hamidi’s device



Context map - version 3 - 2021

Pretty good fit!

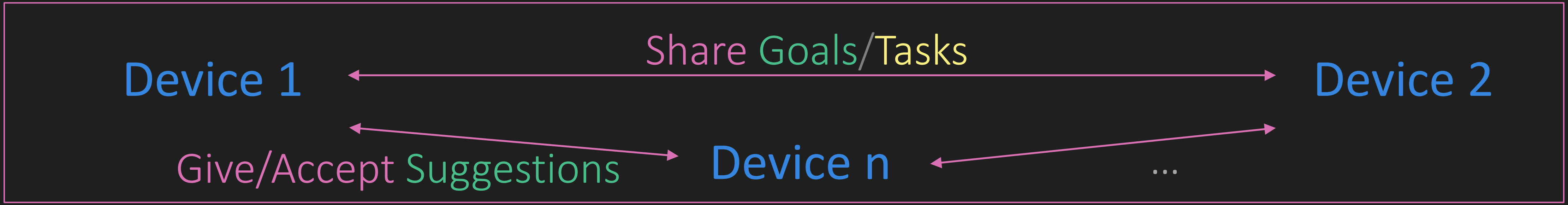
Person's device : fits CRUD

Graph of Goals and Suggestions

Scheduler : fits functional style

Graph of Goals $\longrightarrow$ Tasks

Collaboration : unpredictable interaction => actor/oop model



Added privacy and cost bonus

- All the extra login/cloud stuff we needed
  for people to pay for a central 'all-knowing' coordination point

  ...disappeared!

- Costs per user dropped to near-zero ☺

  A new person's device generates a UUID locally
  and uses that as identity!
  They also run their scheduling algorithms locally ... at no cost to us. ☺

Still, we ‘felt’ issues in the Scheduler domain

```
struct Goal {
    id: Uuid,
    title: String,
    completed: bool,
    start: DateTime,
    deadline: DateTime,
    duration: Duration,
    on_days: Vec<Day>,
    after_hour: Hour,
    before_hour: Hour,
    repeat_interval: Duration,
}
```

$\xrightarrow{\text{Schedule optimization service}}$

```
struct Task {
    goal_id: Uuid,
    title: String,
    start: DateTime,
    end: DateTime,
}
```

This time stuff is getting quite complex…

… so, we 'challenged' the domain expert …

Time constraints ≠ What I want

```rust
struct Budget {
    id: Uuid,
    title: String,
    start: DateTime,
    deadline: DateTime,
    on_days: Vec<Day>,
    after_hour: Hour,
    before_hour: Hour,
    min_per_day: Duration,
    max_per_day: Duration,
    min_per_week: Duration,
    max_per_week: Duration,
}
```

```rust
struct Goal {
    id: Uuid,
    title: String,
    total_duration: Duration,
    duration_left: Duration,
    start: DateTime,
    deadline: DateTime,
    repeat_interval: Duration,
}
```

Surprise!

- We discovered a new domain concept: Budget
  that came from code(rs)
  but was relevant, hidden, *in* the domain

- Now all we had to agree on the name…

We settled on ‘time Budget’ … for now. ☺

Domain from person's view - v4 - 2022

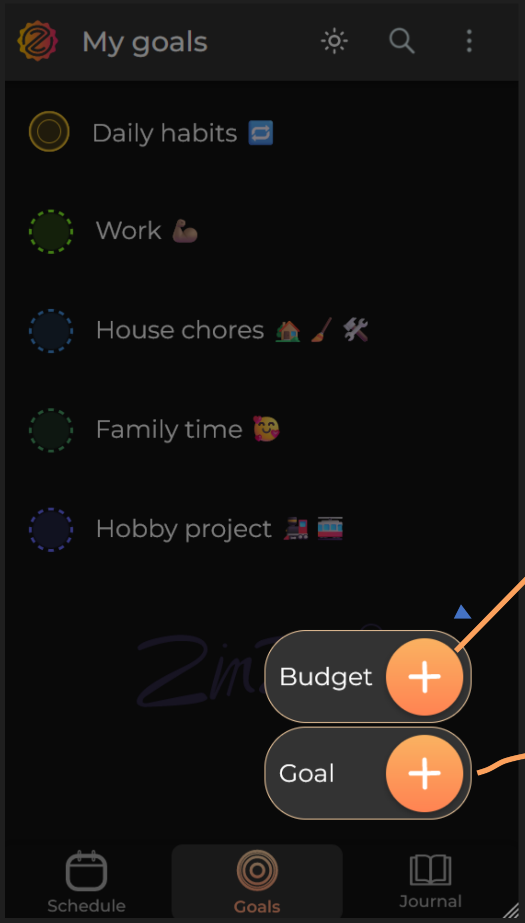
Person's device domain : fits CRUD

Graph of Budgets and of Goals,

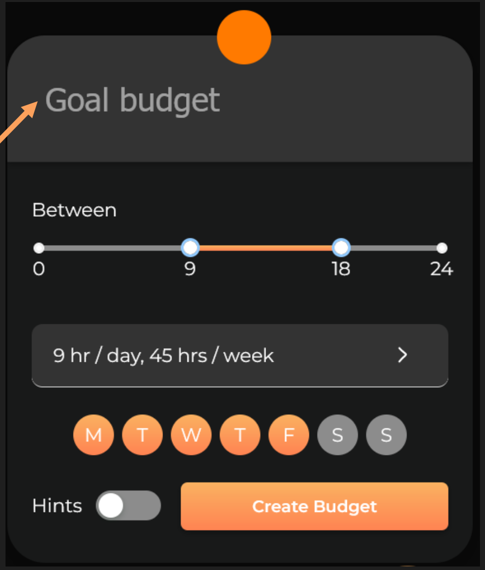
complemented with Suggestions from the Collaboration domain

and Tasks from the Scheduling domain

.. this also simplified our UI/UX ☺


My goals
Daily habits 🔁
Work 💪
House chores 🏠 🧹 🛠
Family time 🥰
Hobby project 🚂 🚃
Budget +
Goal +

- Add a time Budget per 'area' of your life

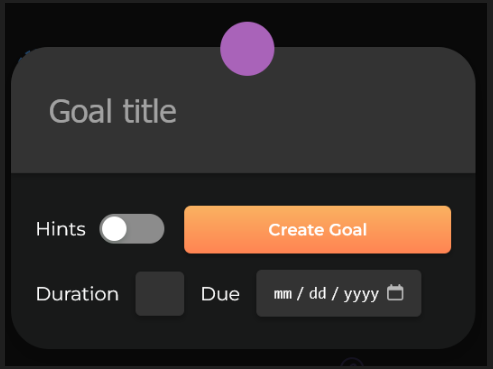

Goal budget
Between

9 hr / day, 45 hrs / week

Hints
Create Budget

Way simpler than GTD!

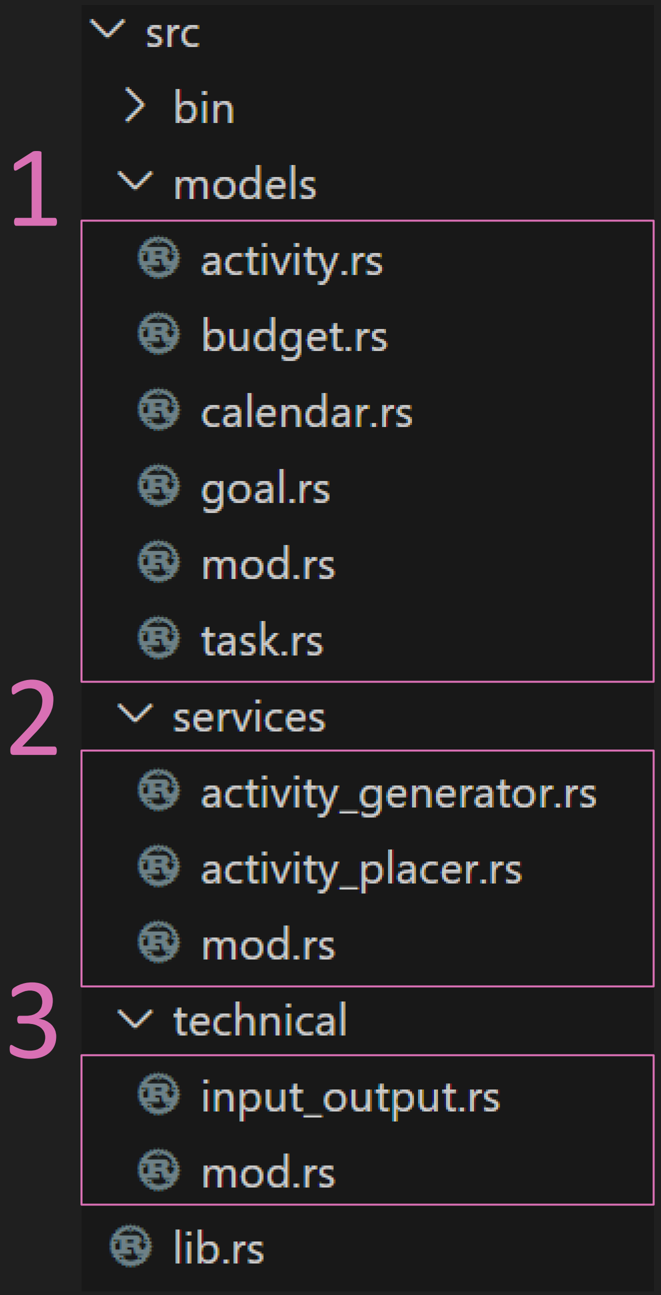
- … and then quick-add Goals with durations


Goal title
Hints
Create Goal
Duration
Due
mm / dd / yyyy

Yes – the design still needs work…

Separating domains has more benefits

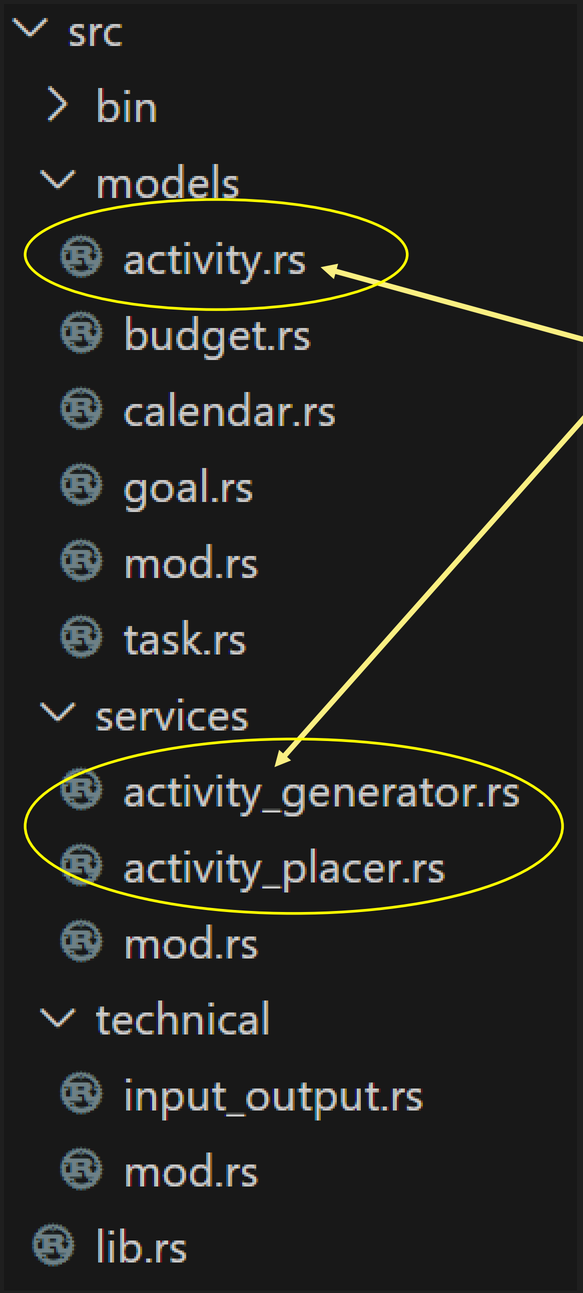
- Extra services
  - specific to the domain
- Extra concepts
  - specific to the domain
- Organize code in modules that ‘explain’ the domain



```
src
  bin
  models
    activity.rs
    budget.rs
    calendar.rs
    goal.rs
    mod.rs
    task.rs
  services
    activity_generator.rs
    activity_placer.rs
    mod.rs
  technical
    input_output.rs
    mod.rs
  lib.rs
```

Modules in the Scheduler

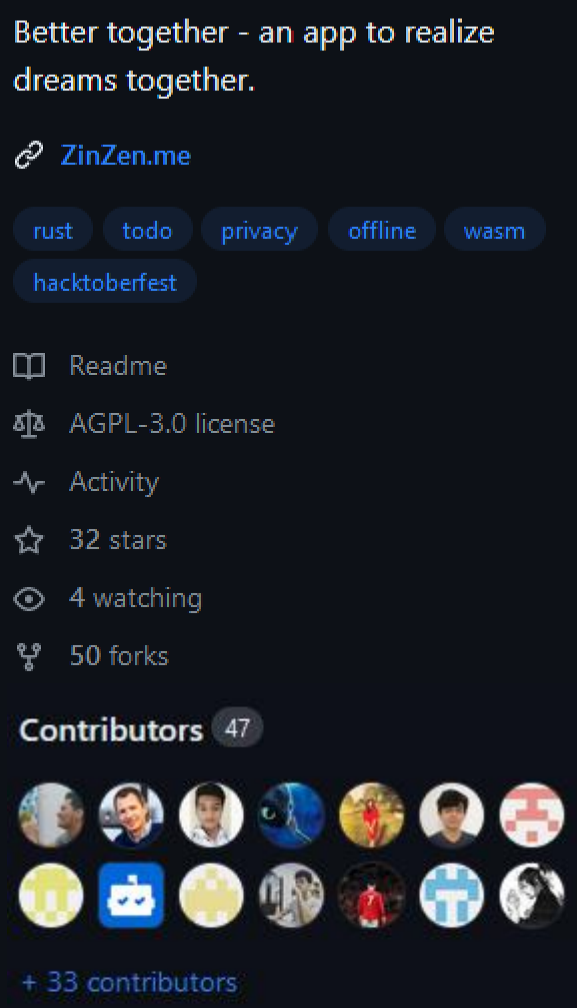
- Modules allow a one-glance overview of the code
- Low coupling, high cohesion
  - Fits in your head
  - One feature, one place
  - Easy to test
- Allow separating technical concerns from domain logic, like:
  - Interaction with file system
  - (de)serializing JSON



```
src
  bin
  models
    activity.rs
    budget.rs
    calendar.rs
    goal.rs
    mod.rs
    task.rs
  services
    activity_generator.rs
    activity_placer.rs
    mod.rs
  technical
    input_output.rs
    mod.rs
  lib.rs
```

Extra concepts and services

- Activity was invented to unify Goals and Budgets for scheduling purposes.
- Activity is only useful in the Scheduler domain, the 'bounded context' of the Scheduler.
- Similarly, Suggestions can't be found here. They are 'bounded' to the Collaboration domain.


Better together - an app to realize dreams together.

ZinZen.me

rust todo privacy offline wasm hacktoberfest

Readme

AGPL-3.0 license

Activity

32 stars

4 watching

50 forks

Contributors 47

+ 33 contributors

Inspired?

Thanks for listening!

ZinZen®

and thanks to all contributors ;)

Credits for pictures via Unsplash

- Mantas Hesthaven
- Barbora Dostálová
- Annie Spratt
- Aziz Acharki

Ideas to possibly expand upon:

- Explaining an aggregate – with invariance boundaries
- Technical pitfalls :
  - testing at the wrong level
- Todo/Task is not an entity – it's a value object
- What domain events do we have?
- Complexity in scheduling domain due to dates
  - Very complicated business logic
  - By separating the code into two steps / modules
    (Activity generator and Activity placer)
    we avoided the mental load/complexity of date-calculations
    – reducing placing to 'does the block size fit in the timeline gap - or not'?