Final Kickstarter App Design Document

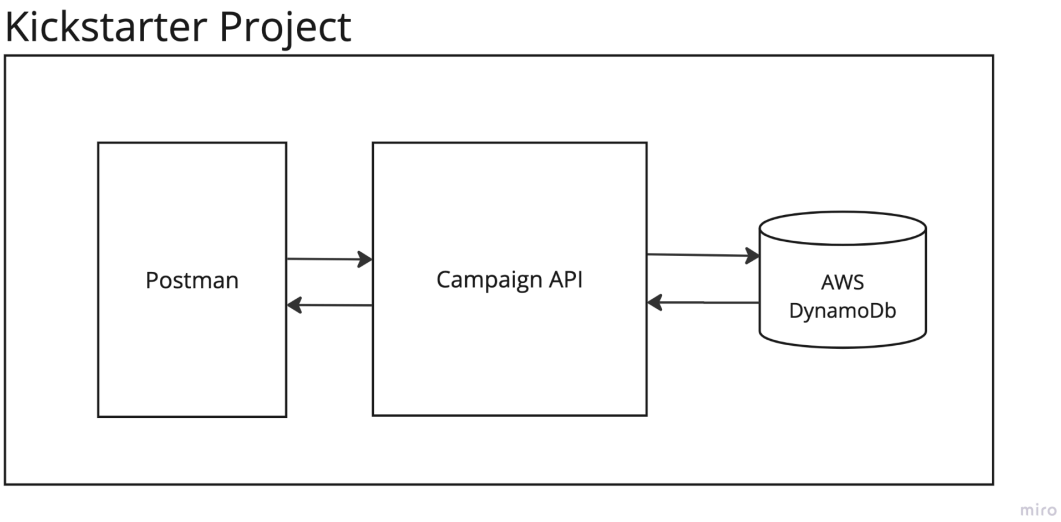
1. Introduction

This document is the condensed version of my original Design Document.
The original document ended up long and it is hard to see what I could have built vs what part I decided on building. This is about **what I ended up implementing**.

2. Design Overview

In my project I chose to focus on the Campaign API portion of the application. In a microservices architecture that could correspond to the Campaign service. My application is small, so it's a monolith.

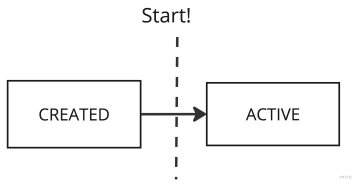
Here is the big picture



2.1. Campaign Lifecycle

In a complete system a campaign would go through many phases:
Pre-funding (when campaign is created)
During funding (when donations are accepted)
After funding (no donations accepted, but voting might take place)
Etc.

Due to time constraints, I implemented the first two stages:



2.2. Assumptions About The Campaign Lifecycle

To make things a bit more interesting, I assumed the following:

2.2.1 CREATED

A campaign is initially created by a user (campaign manager) and before it becomes active, the user may do with this campaign whatever they please - edit most fields (name, description, target_amount, deadlines, etc.).
They can even delete the campaign - all is good.
The campaign manager is the only user who can see campaigns in this state. Other users are not allowed to see this campaign yet.

Once the campaign manager is happy with what they have, they can choose to “Start” the campaign by sending an appropriate API request, opening it for donations. Other users will not be able to make this request successfully.

PUT	/campaigns/{campaign_id}/start	None	None

As soon as the campaign is started it goes to the next state -

2.2.2. ACTIVE

An active campaign is open for donations, but the campaign cannot be changed or deleted by the user at this point.
So, the request to delete or update an active campaign will fail. Obviously, starting an active campaign makes no sense, so that kind of API request will also fail.

All users should be able to see all existing active campaigns in my system.

2.2.3 Implications

Because there are 2 different states, there are 2 different endpoints:

GET	/campaigns	None	[] Campaign
GET	/campaigns/active	None	[] Campaign

The “/campaigns” endpoint allows the user to see **all their campaigns**, regardless of what state they are in.
The “/campaigns/active” endpoint allows any user to see **all active campaigns**, regardless if they are the campaign manager or not.

2.3. Features List

My API allows a user to:

- Create a new campaign
- Start an existing campaign that has not been started yet (other users can’t do that) When it happens, the campaign is marked as ACTIVE and theoretically can accept donations (donations are not part of this project)
- Edit an existing campaign before it is started and marked as ACTIVE (other users can’t do that)
- Delete an existing campaign before it is started and marked as ACTIVE (other users can’t do that)
- View **any** of the campaign that they have created (other users can’t do that)
- View **all** campaigns **they have created** regardless of their state
- View **all active campaigns** regardless who created it

3. Detailed Design

Are you still reading? Great, let’s take a closer look at the endpoints I have implemented.

3.1. Endpoints

Campaign API Endpoints

Verb	URL	RequestBody	Response
POST	/campaigns/create	CreateCampaignRequest	String
PUT	/campaigns/{campaign_id}	UpdateCampaignRequest	None
PUT	/campaigns/{campaign_id}/start	None	None
DELETE	/campaigns/{campaign_id}	None	None
GET	/campaigns/{campaign_id}	None	Campaign
GET	/campaigns	None	[] Campaign
GET	/campaigns/active	None	[] Campaign

All endpoints return 4xx status codes in case of an error or a 200 http status code in case of a success.

3.2. Objects

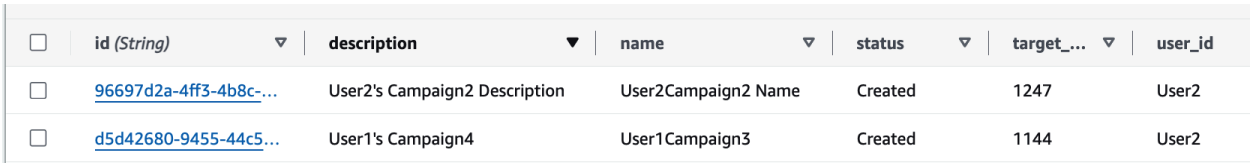
You might be curious about the objects listed above, so here is what they are:

```
pub struct Campaign {
    pub id: Uuid,
    pub user_id: String,
    pub name: String,
    pub description: String,
    pub target_amount: i64,
    pub status: CampaignStatus,
}

pub struct CreateCampaignRequest {
    pub name: String,
    pub description: String,
    pub target_amount: i64,
}

pub struct UpdateCampaignRequest {
    pub name: String,
    pub description: String,
    pub target_amount: i64,
}
```

Notice that Create and Update Campaign Requests look identical right now - this is a decision made to make sure that in the future they can include other fields and become different from each other. Merging them into one now would make those changes difficult in the future.
(OK, I don't plan to work on this specific project in the future, but in real life this is how I usually design things).

3.3. Database

Database consists of just one table - CAMPAIGNS. Since I did not implement authentication, there was not much point in creating a table for the users. The field user_id would be foreign key to that table.
What does this table look like? Like this:



id *(String)*	description	name	status	target_...	user_id
96697d2a-4ff3-4b8c-...	User2's Campaign2 Description	User2Campaign2 Name	Created	1247	User2
d5d42680-9455-44c5...	User1's Campaign4	User1Campaign3	Created	1144	User2

4. Tech Stack

4.1. Frontend

I used Postman to approximate the frontend for my code. You can do the same, see my README for the details.

4.2. Backend

The backend is written in Rust, it includes an http server, handlers, database access layer, tests, etc.
I picked the actix web framework for my http server code because I was somewhat familiar with it. Unfortunately, not familiar enough to implement any authentication or to properly mock the database layer.

4.3. Database

I chose to use AWS DynamoDB for this project for several reasons:

- It might be easier for others to run my project - no need to setup a specific database, you only need to worry about the Rust part
- I have worked with DynamoDB before and am very familiar with it

- It gave me a chance to try a more modern rust library - aws_skd_dynamodb to work with as opposed to what I had to use at my job (rusoto)

Summary

I set out to build my project and I did it!
I can talk more about it, but I hope to do so in person.
This document should be enough to give a good overview - if not - ping me.