Background

Akkroo's software helps businesses collect customer data at many different kinds of events. To be most effective, the data capture software we build must be flexible enough to suit many different use-cases. Data is collected from a consumer in one form or another, processed and then stored ready to use. Building flexible systems is hard, and so we are looking for developers who are up to this challenge.

Your task

- Create a small data capture system that is flexible enough to be able to create many different **processes** from a library of reusable **components**.
- Below you'll find diagrams of two data capture processes, based on real scenarios for our customers which you need to model.
- Your system should be powerful enough to allow it to support both of these processes, **without writing two independent systems**.
- Your system must be flexible enough that brand new processes can be formed using other components without having to make a lot of code changes; effectively a process of completely interchangeable components.

What we've provided you with

- An initial Library of Components (below) that is used in our hypothetical processes. New components might be created in the future that we would want to 'drop into' these processes to change or extend them.
- Some illustrated scenarios that explain how the application should react when it is being used.
- A series of assertions in the code that you can execute/call as a user completes each process-step to check that your system is fulfilling the specification. The components have been lettered so you can match them up with our assertions (see `task_1_validator.js`).
- A written list of assertions that cover how the application should behave visually for the user.
- Sample data for the scenarios inside file `task.js`.
- An example JSON process-definition that may help you think about the design of your own system.

What you should deliver

- A compact JS + JSON implementation
- HTML views for the user interactive parts of the described processes
- (For extra points!) Create an additional process of your own, powered by the system you've written and using any combination of components you like from the library provided.

Should you choose to use a JS framework, you should be prepared to describe your solution in depth in a follow-up discussion with us because we might not be familiar with the framework (we happen to use React JS).

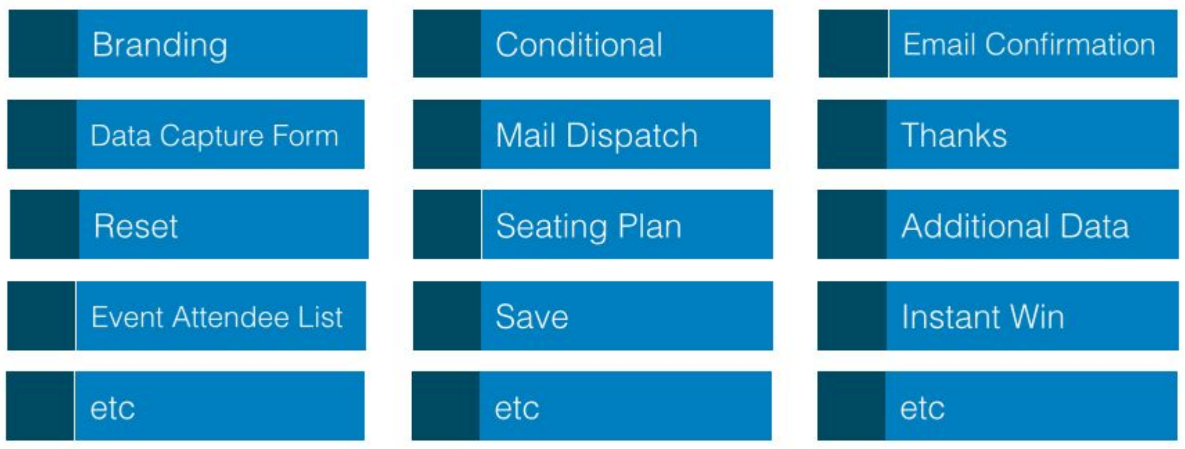
You are free to take the UI design as far as you like, but this is mainly a test of your ability to create abstract concepts that describe real-life processes.

Remember!

At Akkroo we need to be able to design flexible systems. This task is not about building two separate systems that can execute the above processes, it's about demonstrating that you can build a flexible system that can power many processes using interchangeable components.

Library of Components

Here is the component library we'd like you to work with (you *won't* need to build all of these). The processes in the task below are built with components from this library. Review these briefly, then read on!

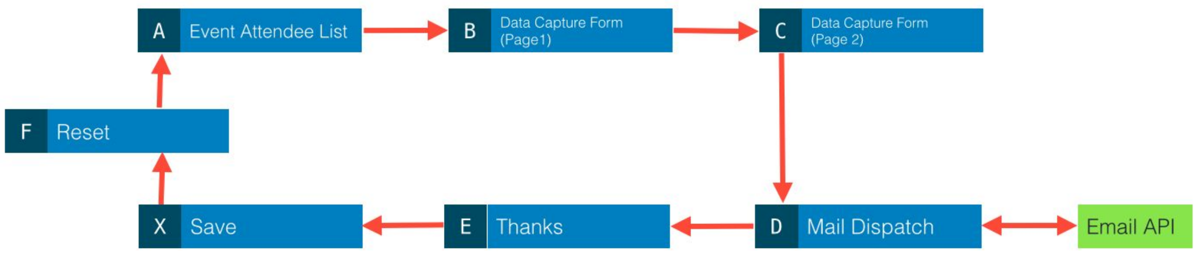


(plus all the other components we might ever build to expand the functionality)

The Task

First Process

Visual description:



Written description:

John Jameson walks over to the exhibition booth and sees the iPad running the software. He selects his name from the Event Attendee List (component A).

He then enters his email address on the first screen (it wasn't present in his record, so wasn't pre-filled) and opts in the receive marketing emails on the second screen. When leaving the booth he receives a marketing email thanking him for attending.

Next, Sam Stuarts walks over to the exhibition booth. He selects his name from the list of event attendee names, enters his email address (as it wasn't present on the initial record) and doesn't opt-in to receive marketing emails. He then leaves the booth and receives no marketing email.

Later on in the afternoon, Matt Michaels arrives, selects his name on the event attendee list, opts-in to receive emails and corrects his email address which was pre-filled as it was stored on his record, but it contained a mistake. He leaves the booth and immediately receives a marketing email thanking him for attending.

In this process, you will need to include the following components from the library:

SOME TITLE	SOME OTHER TITLE
Event Attendee List	A list of the attendees (from the sample data)
Data Capture (Page 1)	A simple form that collects the user's email address
Data Capture (Page 2)	A simple form that collects a postcode, a checkbox which a user ticks if they consent to receive emails, and a field that displays the CustomerID field stored on file
Mail Dispatch	Sends an email to the user if they have opted-in to receive emails
Thanks	Displays a simple "thanks for signing up" message to the user
Save	\`console.log\`s the details captured from/about the user (in a real-world, this component might persist data to a database) ● We would expect to see properties like 'email', 'postcode', 'optedIn' here
Reset	Resets the application ready for the next user

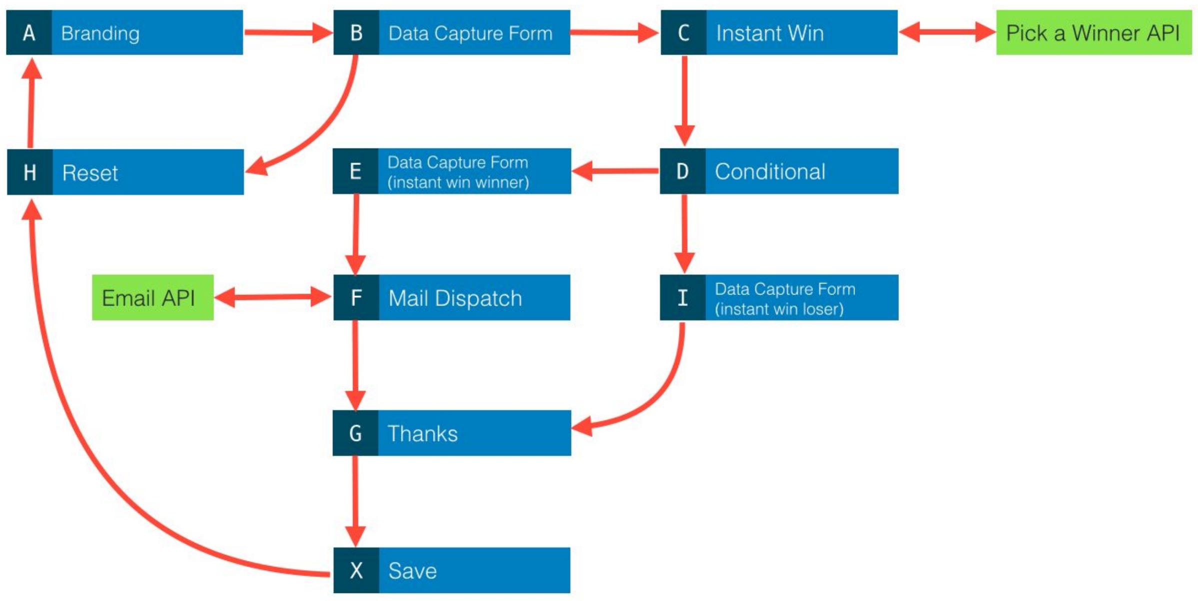
View assertions:

Component	View Requirement
Data Capture (Page 1)	Must contain a form for collecting/editing an email address
Data Capture (Page 2)	Must contain a form for collecting/editing: ● postcode ● checkbox
	Must contain a place to display the CustomerID stored on file
Mail Dispatch	Must have no UI view (behaviour only)
Thanks	Must display a simple “Thanks for signing up” message
Save	Must have no UI view (behaviour only)
Reset	Must have no UI view (behaviour only)

The system must be able to be used as if each scenario happened one after the other in a continuous process. It should not require restarting between each scenario.

Second Process

Visual description:



Written description:

> Alan Adams is approached by a street merchant, offering him a chance to win an incredible prize. He is handed an iPad, clicks the brand's logo (Branding - component A). He enters his name on the next screen and hits 'submit'. Even though he doesn't know it yet, he's won a prize, and the next screen asks him for his email and to select his preferred prize from five options. The following screen then informs him of his lucky win. With a smile on his face he immediately receives an email containing a voucher code which he can use to redeem his prize.
>
> Dennis Davids is then approached by the same street merchant with an offer to win an incredible prize. He is handed an iPad, clicks on the brand's' logo, enters his name and hits 'submit'. He enters his email address on the next page and hits 'submit'. The next screen then notifies him that it's not his lucky day and he hasn't won anything.

In this process, there should be the following component types:

SOME TITLE	SOME TITLE
Branding	A simple logo/image representing a hypothetical consumer brand *e.g. Apple or BMW*. When the user clicks on the image, they will progress to the next component/step
Data Capture Form	A simple data capture form that collects the name from the user

Instant Winner	Calls the “Pick a Winner API” to determine if this user is a “winner”, and if they are, get a voucher code
Conditional	Separates winners and losers into different flows (hint: remember we want flexibility, how would this component be re-used with components other than the Instant Winner one?)
Data Capture Form (instant win: winner)	A data capture form that collects ● email ● preferred prize
Data Capture Form (instant win: loser)	A data capture form that collects an email only
Mail Dispatch	Same component as previous flow, but sends a “Congratulations!” message in the email body
Thanks	Shows a thank you message, and congratulates the user if they are a winner. Displays “Better luck next time!” to any losers.
Save	Same as previous process
Reset	Same as previous process

View assertions:

Component	**View Requirement**
Branding	Must display a logo
	Must move onto the next component by a user-driven click
Data Capture Form	Must collect name
	If the user has not submitted their email within 1 minute, must reset the system ready for another user
Instant Winner	Must have no UI view (behaviour only)
Conditional	Must have no UI (behaviour only)
Data Capture Form (instant win: winner)	Must collect email
	Must collect a prize selection

Data Capture Form (instant win: loser)	Must collect email
Mail Dispatch	Must have no UI (behaviour only)
Thanks	Must display a congratulations message if this user is a winner, and a “Better luck next time!” message if they are a loser
Save	Must have no UI (behaviour only)
Reset	Must have no UI (behaviour only)

The system must be able to be used as if each scenario happened one after the other in a continuous process. It should not require restarting between each scenario - just as the user would experience it.

Important Notes

- You don’t need to write code for the APIs (green boxes in the diagrams), as we have provided functions that mock these calls
- Don’t worry if you get stuck implementing this system, you should still return your ideas. The aim of this exercise is to be able to discuss your decision-making process, not only assess your coding skill.
- Remember that you can reach out to us if you need more explanation/help on this task!