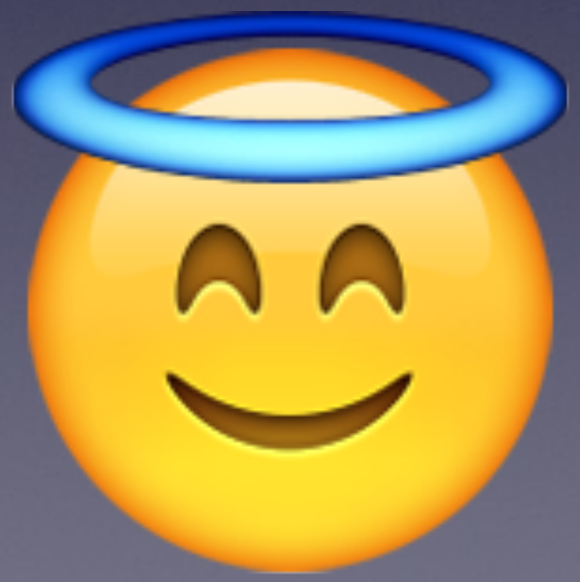
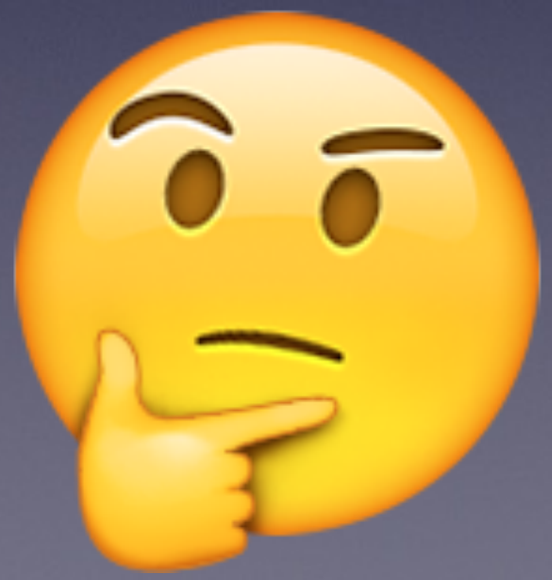
React Conformist

A Flexible Approach to Forms

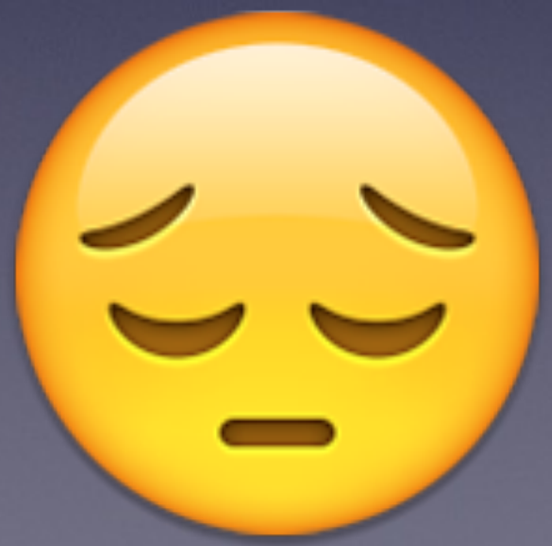


Let’s talk about forms.

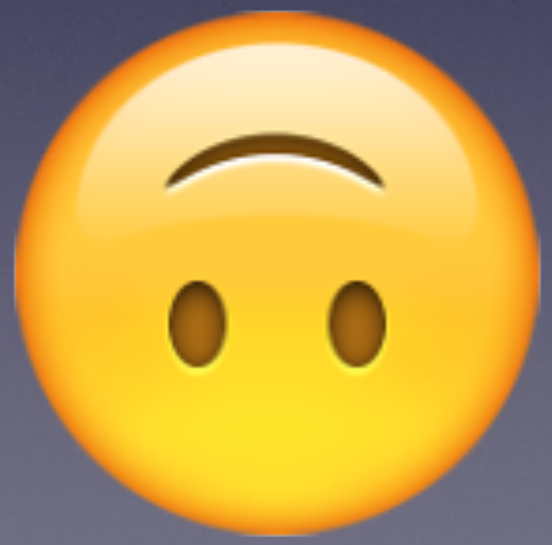
Seems like they should be simple…



If you've ever made a form, you know this to be false.

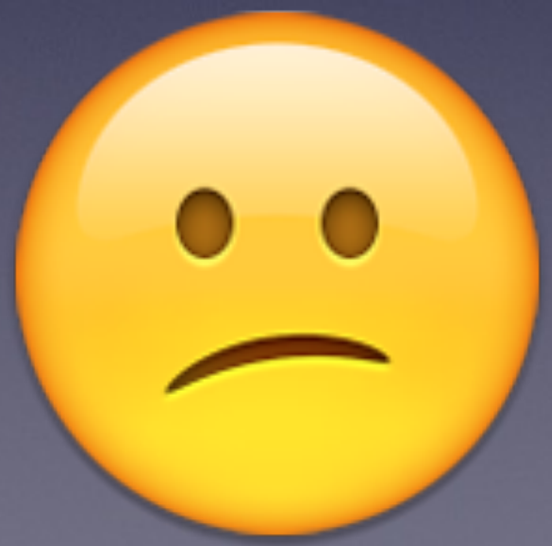


We need a flexible approach!

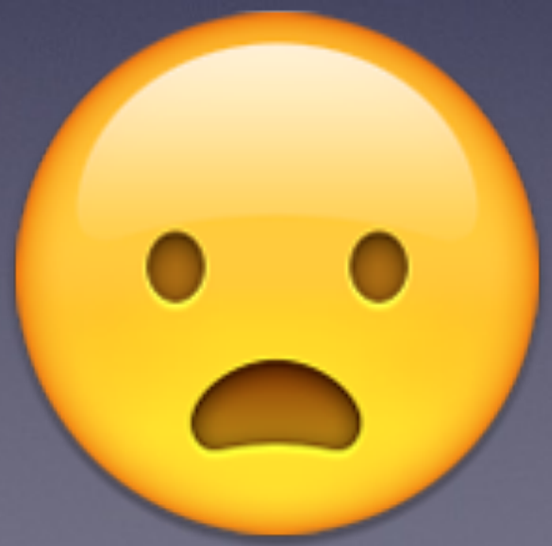


What doesn’t work?

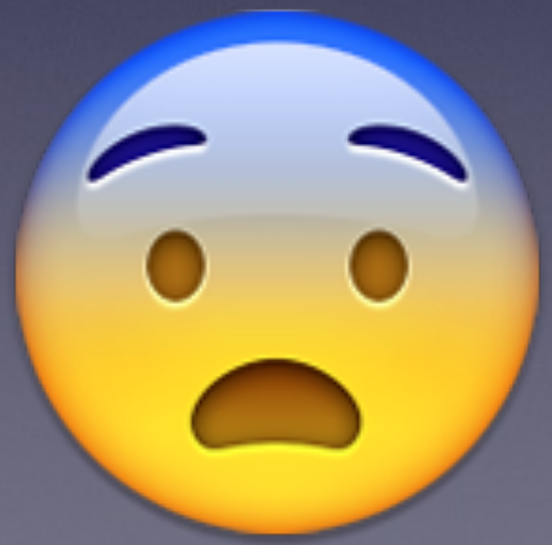
Define a static mapping from schema to UI



Provide only one component for e.g. Form, Field, Input



Validation policy is defined by the form library

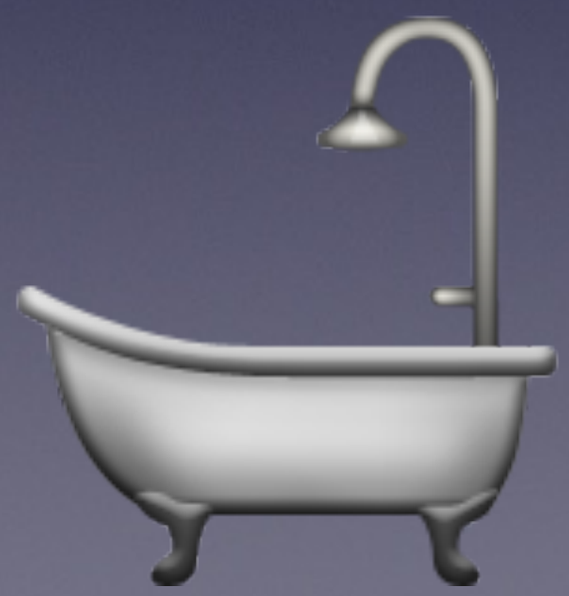


Every one of these assumptions will fail for a large enough app.

Form validation at the core

1. describe your schema

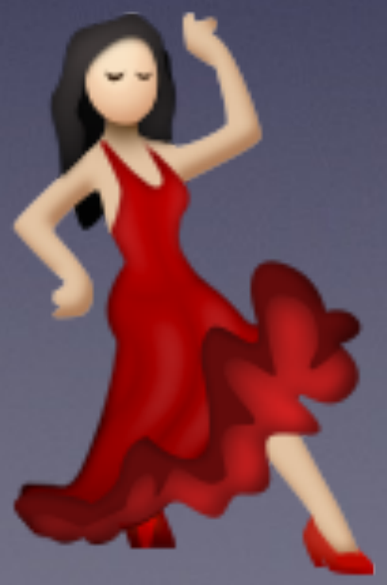
2. pipe data through the validation function



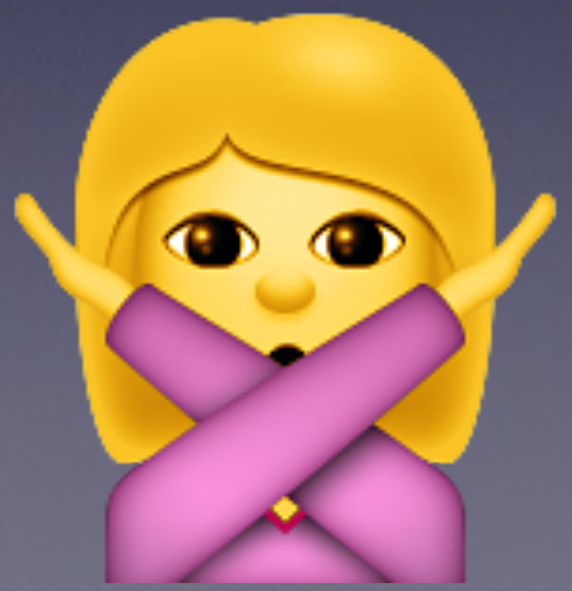
3. ~~profit~~display results!

Where does it all go wrong?

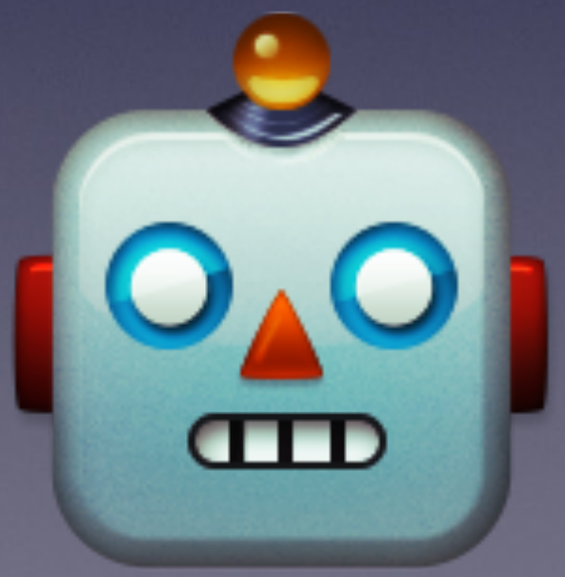
Live validation



When do I validate / show feedback?



How to show server-side errors?

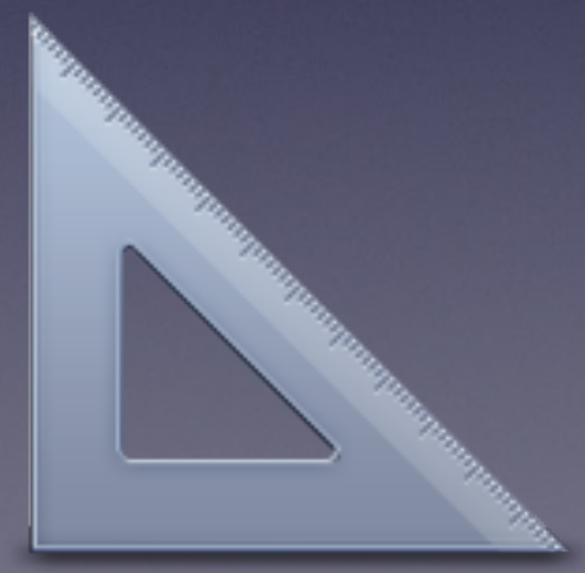


Enough problems.

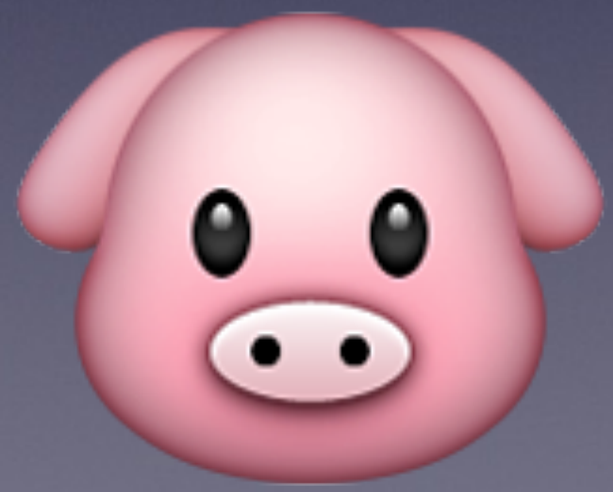
More solutions!

Separate validation mechanics from UI concerns

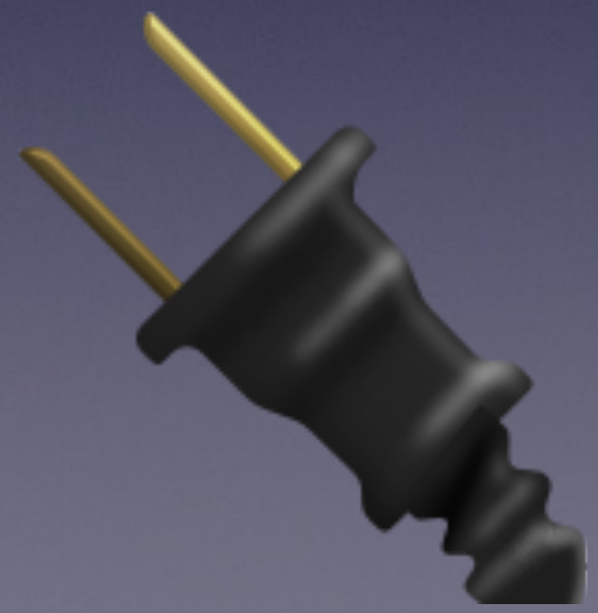
Schema: use conformist



Define UI using plain old react components (PORCs)

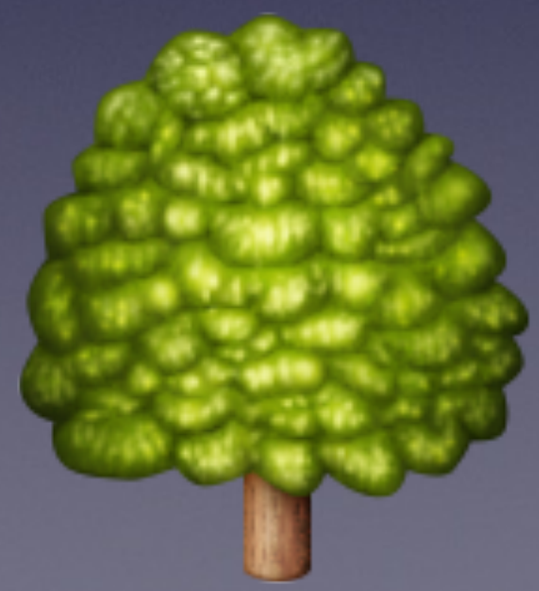


Connect the schema to the UI with form connectors!

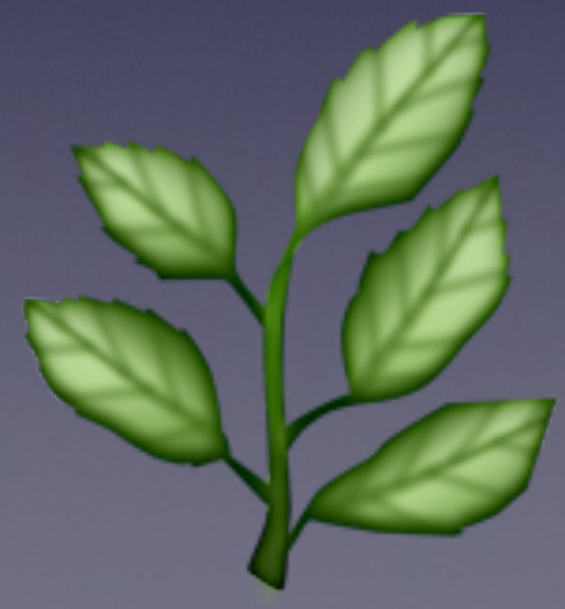


What are form connectors?

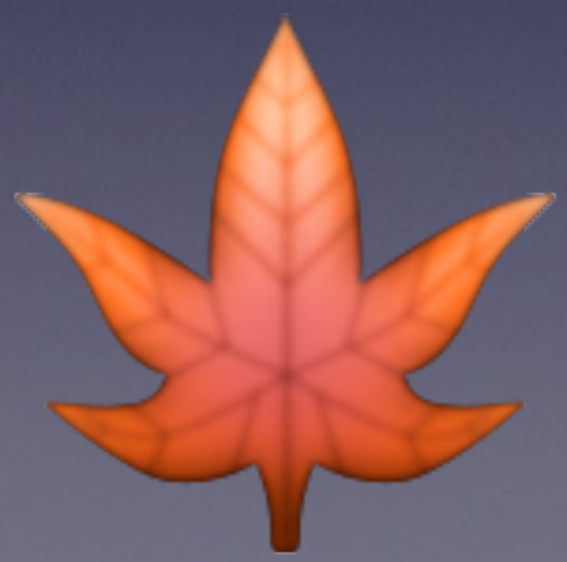
Form Connector is the root of the form: the join point for the schema and errors with form submission



Field Connector maps a UI region to a subsection of the Form schema (nestable for complex schemas)



Input Connector wires up a field to an input, operates in terms of value



Form Connector

field

field

Field Connector

subfield

subfield

field

Field Connector

Input Connector

subfield

Input Connector

subfield

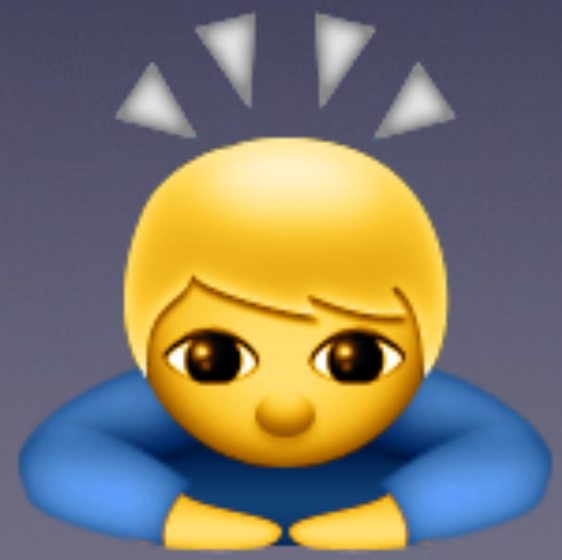
¡CODE EXAMPLES!

Benefits

Schema definition is simple and easy to understand

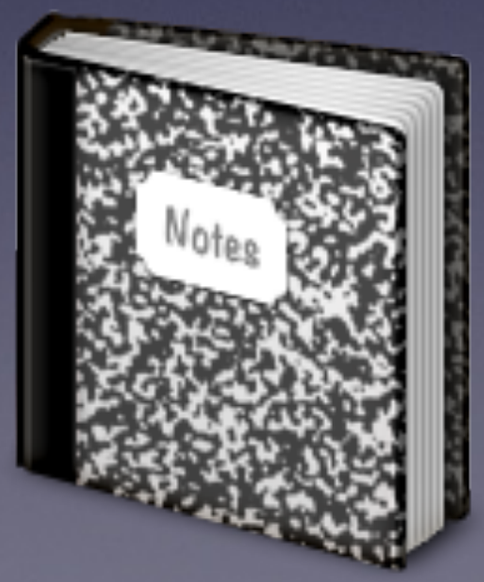
UI Components are easy to test and prototype by passing props

Developer is in control of validation policy and display

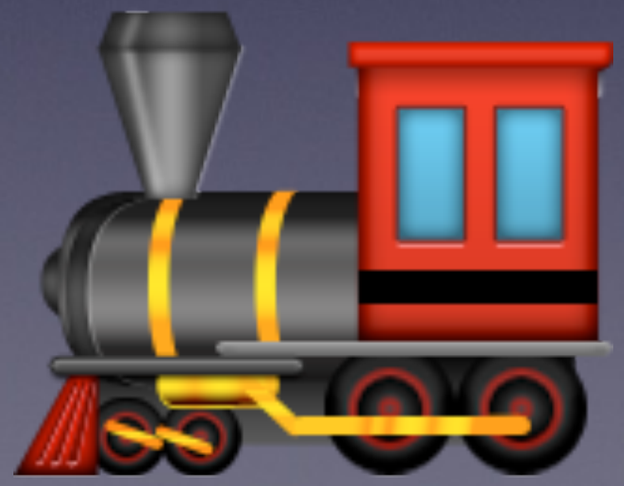


TODO

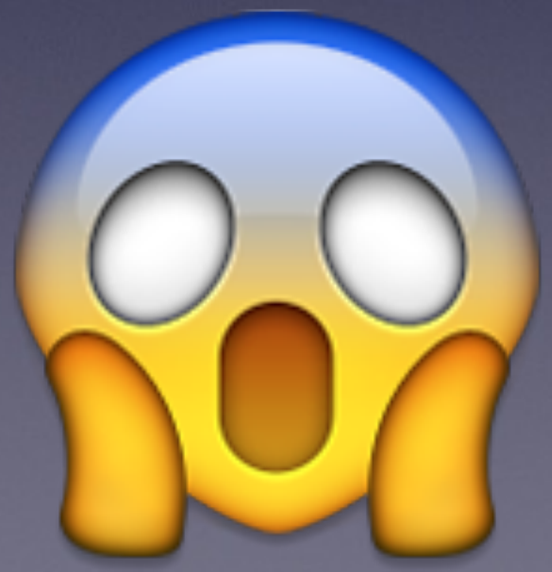
Documentation



Boilerplate Reduction



Publish to npm



Expose more hooks

QA?

Thank you!