Becoming a Form Wizard

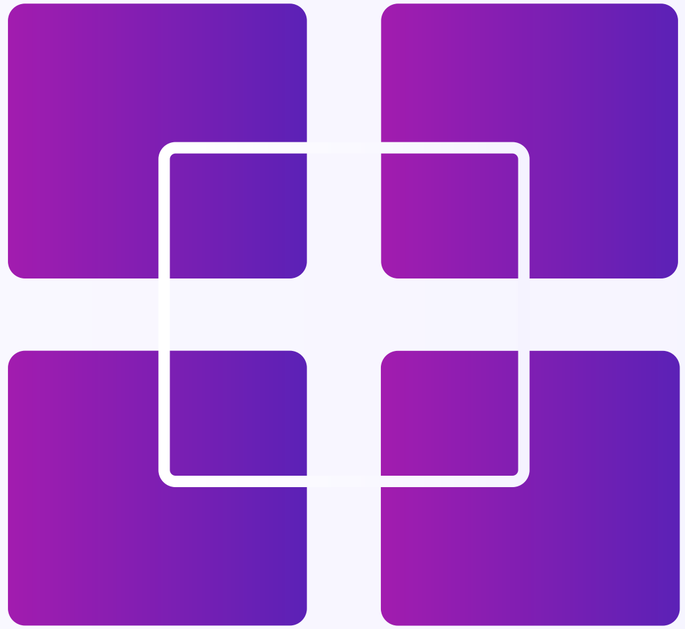
Intuitive Multi-Step Workflows w/ State Machines

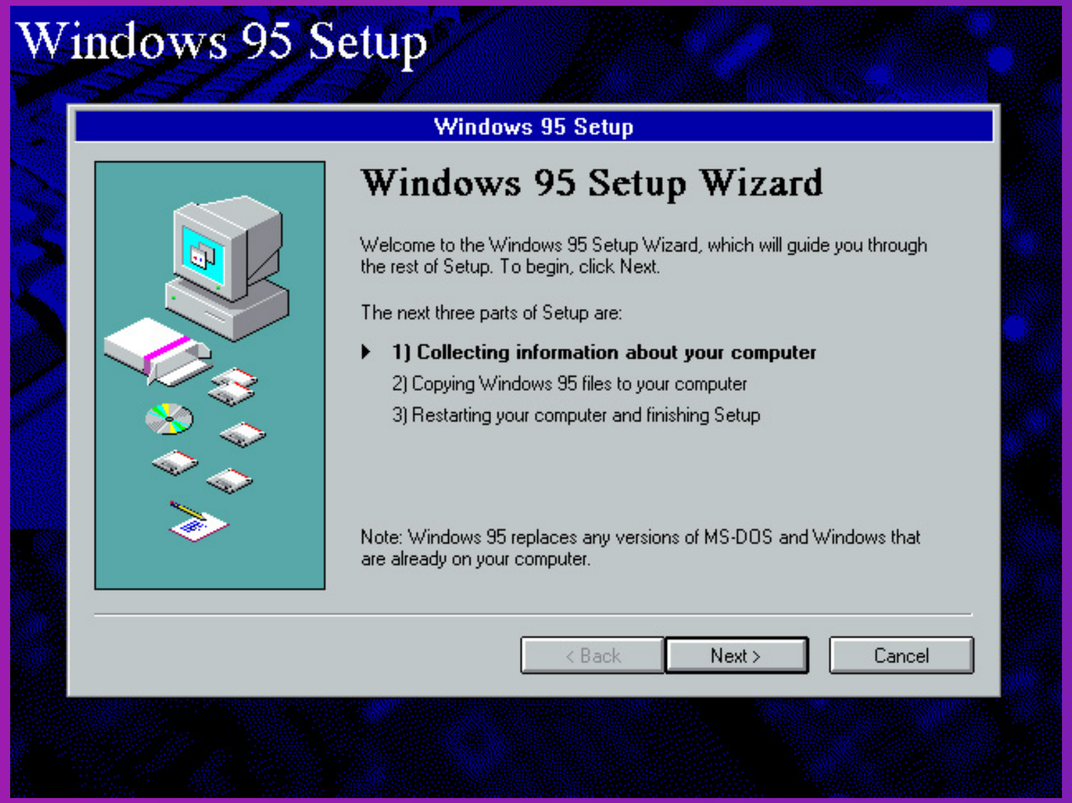
The bit about me

- Nick Hehr
- Staff Software Engineer, Front End Platform Team @Betterment
- Empathetic Community Organizer
- Amateur Embedded JS Developer
- Outdoor cyclist, Indoor climber

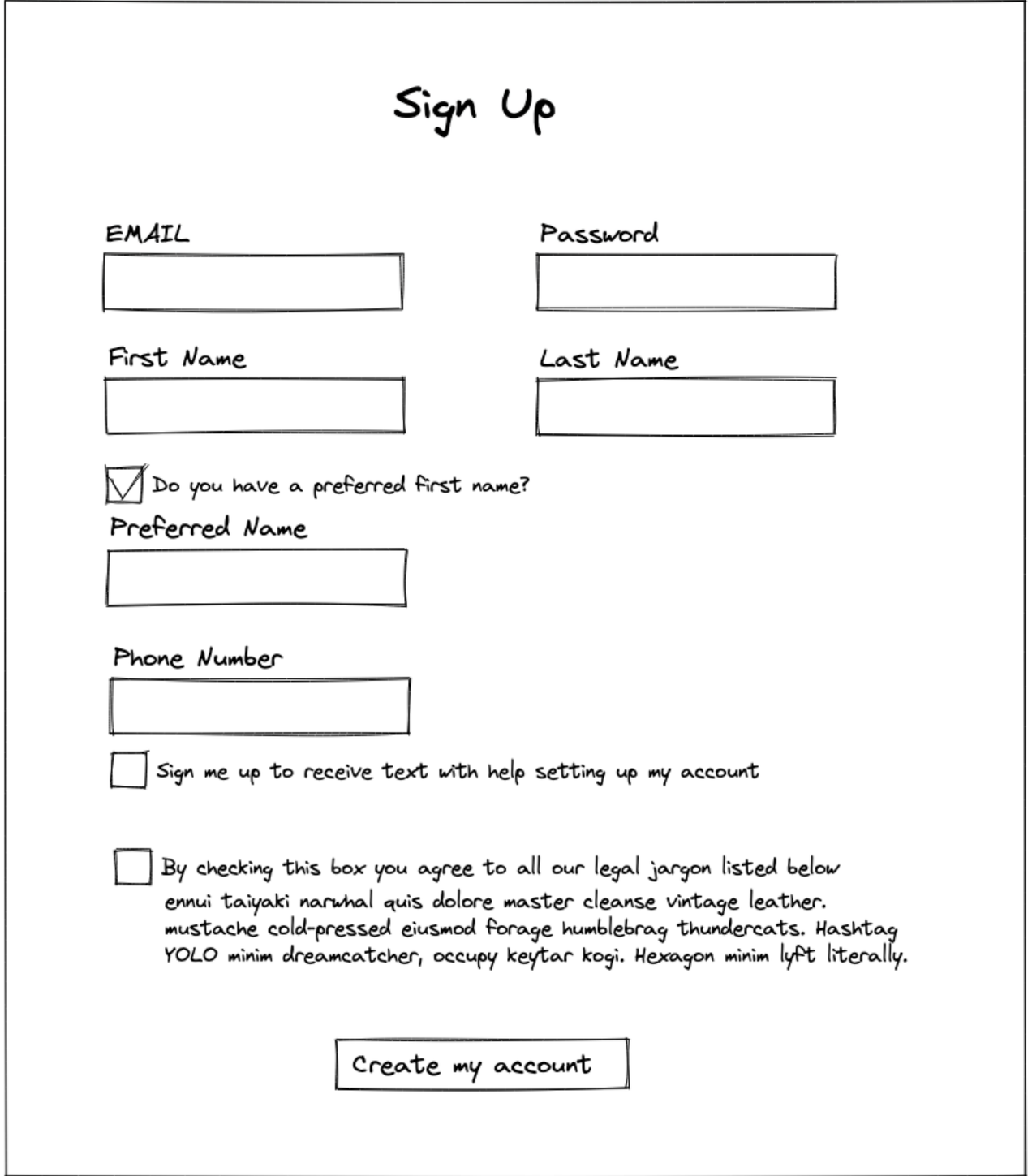
Our journey

- What is a wizard?
- How are they typically created?
- Where do state machines come in?
- What can we use today?
- What lies ahead?





What is a wizard?



What we don’t want

- overwhelming
- distracting
- tedious

How are these experiences typically built?

All regions ▾ Safe search: moderate ▾ Any time ▾

https://blog.devgenius.io › create-a-multi-step-form-with-reactjs-322aa97a2968

Create a Multi-Step Form with ReactJS | by Megan Lo | Dev ...

npx create-**react**-app **multi-step-form**-tutorial After the **React** app has installed, run cd **multi-step-form**-tutorial to access the folder first. Then run code . to open the folder in your preferred code editor. And then run yarn start or npm start to start the server. This is what you see, right? Let's keep the page on and go back to our code.

https://dev.to › sametweb › how-to-create-multi-step-forms-in-react-3km4

How to create multi-step forms in React? - DEV Community

React Step Builder allows you to combine states of multiple components in one place and navigate between **step** components without losing the state from other **step** components. This guide is using **React Step** Builder v2.0.11. If you are using the version v3+, please see the latest documentation on package's NPM page.

https://medium.com › how-to-react › create-multi-step-form-in-react-with-validation-4ac0...

Create multi step form in React with validation. | by ...

7. Now open the StepTwo.js file and paste the below code inside that. 8. Now it's time to show all the values on our final file that's Final.js. Paste the below code inside that. That's it, this...

https://www.digitalocean.com › community › tutorials › how-to-create-multistep-forms-wi...

How To Create Multistep Forms With React and Semantic UI ...

The **multistep form** is using the switch statement, which reads the **step** from state and uses this to select which components are rendered at each **step**. At **Step** 1, the UserDetails component will be rendered. At **Step** 2, the PersonalDetails component will be rendered. At **Step** 3, the Confirmation component will be rendered.

https://reactjsexample.com › react-multi-step-form

React Multi-Step Form - React.js Examples

FORMIK

```jsx
import React, { useState } from 'react';
import { ErrorMessage, Field, Form, Formik } from 'formik';
import * as Yup from 'yup';
import { Debug } from './Debug';

const sleep = ms => new Promise(resolve => setTimeout(resolve, ms));

// Wizard is a single Formik instance whose children are each page of the
// multi-step form. The form is submitted on each forward transition (can only
// progress with valid input), whereas a backwards step is allowed with
// incomplete data. A snapshot of form state is used as initialValues after each
// transition. Each page has an optional submit handler, and the top-level
// submit is called when the final page is submitted.
const Wizard = ({ children, initialValues, onSubmit }) => {
  const [stepNumber, setStepNumber] = useState(0);
  const steps = React.Children.toArray(children);
  const [snapshot, setSnapshot] = useState(initialValues);

  const step = steps[stepNumber];
  const totalSteps = steps.length;
  const isLastStep = stepNumber === totalSteps - 1;

  const next = values => {
    setSnapshot(values);
```

```
import React from 'react';
import { BrowserRouter as Router, Route, withRouter } from 'react-router-dom';
import { Formik, Field, ErrorMessage } from 'formik';
import { Debug } from './Debug';

// the root path. locations should extend from this
const formRootPath = '/step';

// the specific path for each page
const locations = ['/step/1', '/step/2'];

const sleep = ms => new Promise(resolve => setTimeout(resolve, ms));

const required = value => (value ? undefined : 'Required');

class WizardBase extends React.Component {
  static Page = ({ children }) => children;

  constructor(props) {
    super(props);
    this.state = {
      values: props.initialValues,
    };
  }
```

```
import React, { Fragment } from 'react';
import {
  BrowserRouter as Router,
  Route,
  Switch,
  Link,
  Redirect,
} from 'react-router-dom';
import { Field, ErrorMessage, withFormik } from 'formik';
import { Debug } from './Debug';

const required = value => (value ? undefined : 'Required');

const Page1 = () => (
  <Fragment>
    <div>
      <label>First Name</label>
      <Field
        name="firstName"
        component="input"
        type="text"
        placeholder="First Name"
        validate={required}
      />
```

Primary Directives

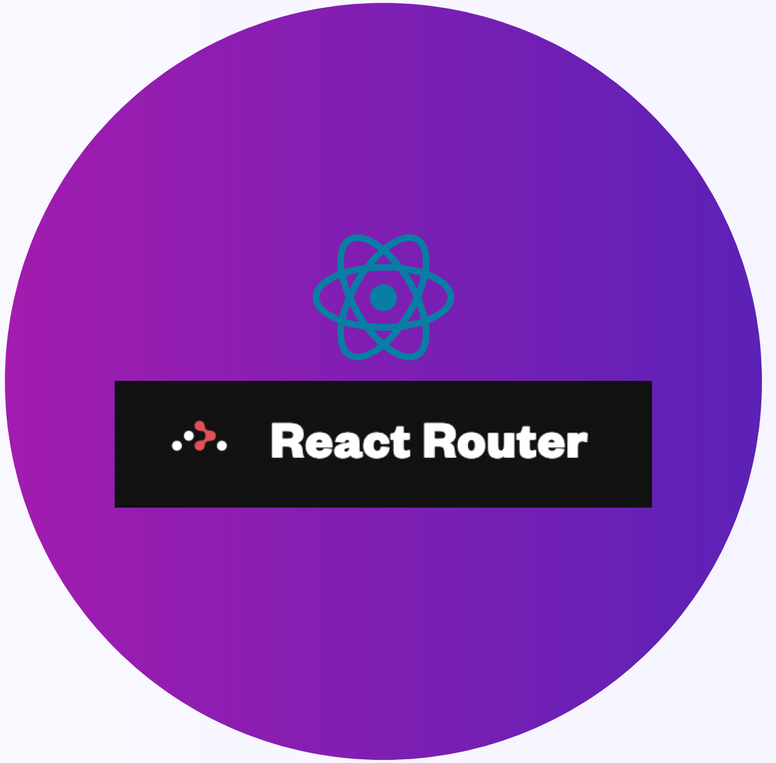
1. Flows will be routed, meaning visible steps should match a path in the URL
2. Navigation among steps should have a single source of truth
3. Flows are form-agnostic, however they can integrate cleanly with Formik as the preferred form library

Dream API

```
<Wizard>
  <Step name="first-name" component={FirstNameStep} />
  <Step name="last-name" component={LastNameStep} />
  <Step name="success" component={SuccessStep} />
</Wizard>
```

Supplies

- React Context
- `useReducer` hook
- React-Router Route components
- custom hook



React Context

```
const WizardContext = createContext(undefined);
```

Provide some context

```
function Wizard({ children, initialValues }) {
  const steps = React.Children.map(children, (result, child) => {
    if (child?.type === Step) return child.props.name;
    return false;
  }).filter(Boolean);
  const defaultStep = steps[0];
  const value = useReducer(wizardReducer, {
    currentStep: defaultStep,
    values: initialValues,
    steps
  });

  return (
    <WizardContext.Provider value={value}>
      {children}
      <Switch>
        <Redirect from="/" to={defaultStep} exact />
      </Switch>
    </WizardContext.Provider>
  );
}
```

Reducer function

```
const wizardReducer = (state, action) => {
  switch (action.type) {
    case 'sync':
      return {...state, currentStep: action.step };
    case 'updateValues':
      return {
        ...state,
        values: {
          ...state.values,
          ...action.values,
        },
      };
    default:
      return state;
  }
};
```

React-Router Route components as steps

```
function Step ({ component: Component, name }) {
  const state = useWizardContext();
  return <Route path={name} render={() => <Component {...state} />} />;
}
```

Custom hook

```
function useWizardContext() {
  const [wizardState, dispatch] = useContext(WizardContext);
  const history = useHistory();

  const sync = useCallback(
    (step) => {
      dispatch({ type: 'sync', step });
    },
    [dispatch]
  );

  const updateValues = useCallback(
    (values>) => {
      dispatch({ type: 'updateValues', values });
    },
    [dispatch]
  );

  const goToStep = useCallback(
    (step, values) => {
      if (values) updateValues(values);
      sync(step);
      history.push(step);
```

Usage

```
<Wizard>
  <Step name="first-name" component={FirstNameStep} />
  <Step name="last-name" component={LastNameStep} />
  <Step name="success" component={SuccessStep} />
</Wizard>
```

Form agnostic

```
const FirstNameStep = ({ goToNextStep, goToPreviousStep, wizardState }) => {
  function handleSubmit(event) {
    event.preventDefault();
    const values = Object.fromEntries(new FormData(event.target));
    goToNextStep(values);
  }
  return (
    <form onSubmit={handleSubmit}>
      <label id="name-label" htmlFor="firstName">
        First Name
      </label>
      <input
        type="text"
        id="firstName"
        name="firstName"
        aria-labelledby="name-label"
        defaultValue={wizardState.values.firstName}
      />
      <div className="actions">
        <button type="button" role="link" onClick={goToPreviousStep}>
          Previous
        </button>
        <button type="submit">Continue</button>
      </div>
    </form>
  );
```

Formik Usage

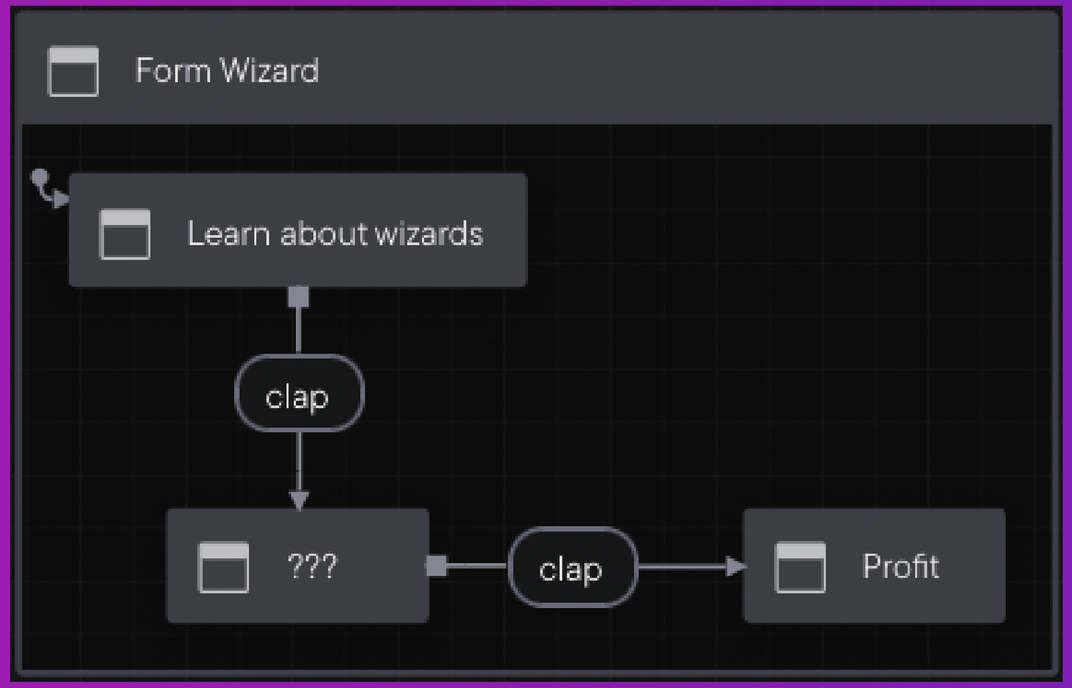
```
const FirstNameStep = ({ goToNextStep, goToPreviousStep, wizardState }) => {
  return (
    <Formik onSubmit={goToNextStep} initialValues={wizardState.values}>
      <Form>
        <label id="name-label" htmlFor="firstName">
          First Name
        </label>
        <Field type="text" name="firstName" id="firstName" aria-labelledby="name-label" />
        <div className="actions">
          <button type="button" role="link" onClick={goToPreviousStep}>
            Previous
          </button>
          <button type="submit">Continue</button>
        </div>
      </Form>
    </Formik>
  )
};
```

No Form Needed

```
const SuccessStep = ({ wizardState }) => {
  const { firstName, lastName } = wizardState.values;
  return (
    <section>
      <h1>Welcome, {firstName} {lastName}!</h1>
    </section>
  )
}
```

Only one way through

```
<Wizard>
  <Step name="first-name" component={FirstNameStep} />
  <Step name="last-name" component={LastNameStep} />
  <Step name="success" component={SuccessStep} />
</Wizard>
```

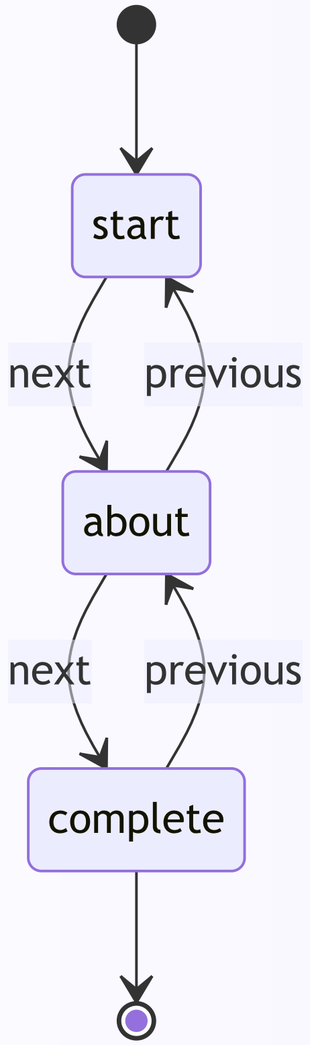


How do state machines help solve this problem?

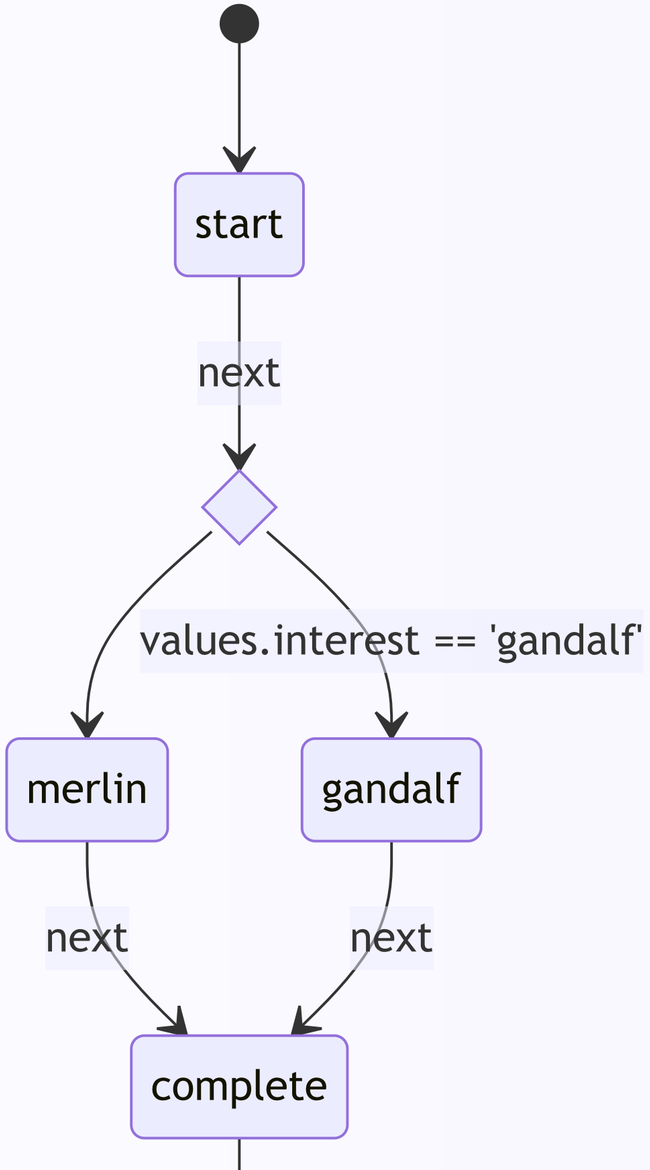
> an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some external inputs; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the conditions for each transition.

"Finite-state machine" page on Wikipedia

an ~~abstract machine~~ **interactive experience** that can ~~be in~~ **display** exactly one of a finite number of ~~states~~ **steps** at any given time. The ~~FSM~~ **wizard** can change from one ~~state~~ **step** to another in response to some external inputs; the change from one ~~state~~ **step** to another is called a ~~transition~~ **navigation**. ~~An FSM~~ A ~~flow~~ **wizard** is defined by a list of its ~~states~~ **steps**, its initial ~~state~~ **step**, and the conditions for each ~~transition~~ **navigation**.


start
next
previous
about
next
previous
complete

How can we control the conditional navigation?


start
next
values.interest == 'gandalf'
merlin
gandalf
next
next
complete

```
const MyWizard = () => {
  const values = {
    name: '',
    email: '',
    interest: '',
  };
  const isInterestedInGandalf = (currentValues, nextState) => nextState.values.interest === 'gandalf';

  return (
    <Wizard initialValues={values}>
      <Step name="start" component={StartStep} next={[['gandalf', when(isInterestedInGandalf)], 'merlin']} />
      <Step name="merlin" component={MerlinStep} next="complete" />
      <Step name="gandalf" component={GandalfStep} previous="start" />
      <Step name="complete" component={CompleteStep} />
    </Wizard>
  )
}
```

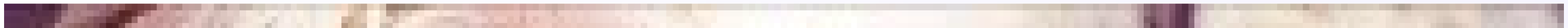
What can we use today?

```
import { createWizard } from 'robo-wizard';

const wizard = createWizard(['start', 'about', 'complete'], { firstName: '', lastName: '' });
wizard.start(updatedWizard => { console.log('Updated!', updatedWizard.currentStep), updatedWizard.currentValues });

console.log(wizard.currentValues); // { firstName: '', lastName: '' }
console.log(wizard.currentStep); // start

wizard.goToNextStep({ values: { firstName: 'Jane' } });

console.log(wizard.currentValues); // { firstName: 'Jane', lastName: '' }
console.log(wizard.currentStep); // about

wizard.goToNextStep({ values: { lastName: 'Doe' } });

console.log(wizard.currentValues); // { firstName: 'Jane', lastName: 'Doe' }
console.log(wizard.currentStep); // complete
```

```
const wizardMachine = createMachine({
  id: 'wizard',
  initial: 'start',
  context: {
    firstName: '',
    lastName: ''
  },
  states: {
    start: {
      on: {
        next: {
          target: 'about',
          actions: ['updateValues'],
        }
      }
    },
    about: {
      on: {
        next: {
          target: 'complete',
          actions: ['updateValues']
        },
        previous: 'start'
      }
```

Let’s check out a demo

Where do we go from here?



How do we communicate about wizards better?

- Visualize wizard behavior
- Iterate with a shared language that Product, Design, and Engineering can all understand.

What are we missing?

- entry / exit steps
- final steps
- branching flows (nested services) for efficient reuse
- server-driven transitions

Thank you & happy building!