React Basics: A Guided Tour with `react_survey_app`

Welcome! In this mini-lecture, we'll introduce the core concepts of React by exploring the real code in your `react_survey_app`. You'll see how components, props, state, and events work together to build an interactive survey.

What is React?

- A JavaScript library for building user interfaces
- Developed by Facebook
    - Open-source and widely used
- Lets you compose complex UIs from small, reusable pieces called **components**
- Component-based, declarative, and efficient

Why Use React?

- Makes building interactive UIs easier
- Large ecosystem and community

Components: The Building Blocks

- Each component does a small thing
- Components can be nested: one can include another

** Nesting in our survey app **

- `SurveyPage` is a top-level component. Includes:
    - `Question`
    - `Prediction`
    - `Feedback`

Elements of a Component

A React component typically has:

- **Props**: Inputs to the component (read-only)
- **State**: Data managed within the component (can change)
- **Event Handlers**: Functions that respond to user actions (like clicks)
- **Rendering**: JSX that defines what HTML to produce

JSX Syntax (Quick Intro)

React uses **JSX** (JavaScript XML)

- HTML-like syntax
- Can "include" other React components

From: `SurveyPage.tsx`

```
<div className="poll-container">
  <section>
    <Question ... />
  </section>
  <section>
    <Prediction ... />
  </section>
  ...
</div>
```

Props: Passing Data to Components

- **Props** are inputs to components (read-only)
- Think of props as arguments to a function

Example Passing props from `SurveyPage` to `Question`

```
<section>
    <Question
      fruits={fruits}
      ...
    />
  </section>
```

State: Remembering Things

- **State** is data managed within a component (can change)
- The `useState` hook creates state variables
  - creates a state variable *and* a setter function to update it
  - we will pass the setter function as a prop so child components can update state

Example from : `src/survey/page/SurveyPage.tsx`

```
const [selectedFruit, setSelectedFruit] = useState<string | null>(null);
const [prediction, setPrediction] = useState<number | null>(null);
```

Handling Events

- React lets you respond to user actions (like clicks) with event handlers

Example: Handling a button click in `Question.tsx`

- `setSelectedFruit` is a function passed as a prop from `SurveyPage`
- When a button is clicked, it calls `setSelectedFruit`, passing in the the selected fruit

```
<button
  type="button"
  className={selectedFruit === fruit ? "selected" : ""}
  onClick={() => setSelectedFruit(fruit)}
>
  {fruit}
</button>
```

Changing the State

```
const [selectedFruit, setSelectedFruitState] = useState<string | null>(null);
const [prediction, setPrediction] = useState<number | null>(null);

// When fruit changes, set the fruit state and reset prediction state
const handleSetSelectedFruit = (fruit: string) => {
  setSelectedFruitState(fruit);
  setPrediction(null);
};


...
  <Question
    fruits={fruits}
    selectedFruit={selectedFruit}
    setSelectedFruit={handleSetSelectedFruit}  // pass the handler
```

Re-rendering

- When state changes, React re-renders the component and its children
    - You don't specify how to update the DOM (page contents)
    - You just specify what should be on the page, based on state/props
    - React figures out what is different and updates that

Principles of State Management

- Lift state to the highest component that needs it
    - In our app, the fruit and the prediction state are in `SurveyPage` because multiple child components need to access them
- Pass state down as props to child components
- Pass setter functions down as props to allow children to update state

Conditional Rendering

- Use JavaScript expressions in JSX to show/hide parts of the UI

Example from your app: Only show the prediction section if a fruit is selected

```
{selectedFruit && (
  <section>
    <PredictionContainer ... />
  </section>
)}
```

Component-specific CSS

- Each component can have its own CSS file for styles
- Example: `Question.css` styles the `Question` component
- As compared to one giant .css file, this keeps styles modular and easier to find and manage

React Developer Tools Demo

- Inspect component tree
- View props and state as you interact
- Edit state directly to see effects

Try It Yourself!

- Explore the files in `src/survey/` to see how the app is built
- Make a small change (like editing a label or adding a new fruit) and see what happens!

Key Files to Explore

- `SurveyPage.tsx`: Main layout and composition
- `Question.tsx`: The question and fruit selection
- `Prediction.tsx`: The prediction input
- `Feedback.tsx`: The feedback display