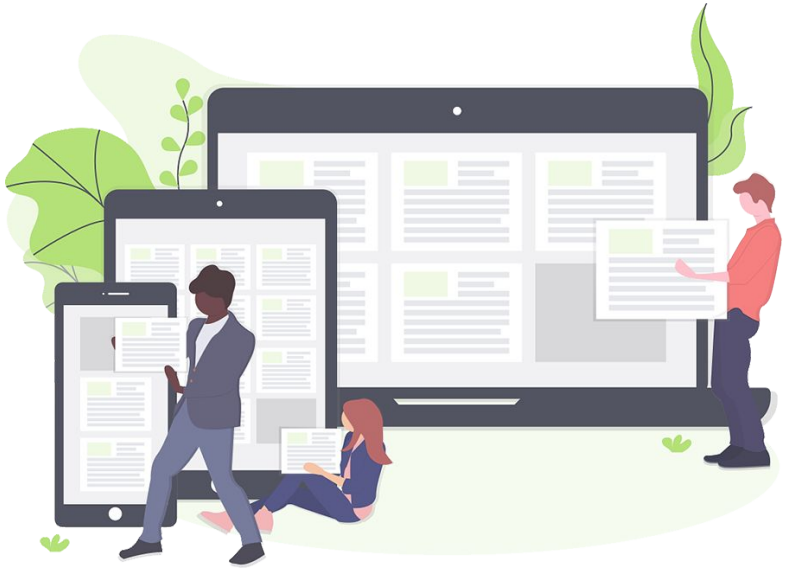
UCLA TeachLA:

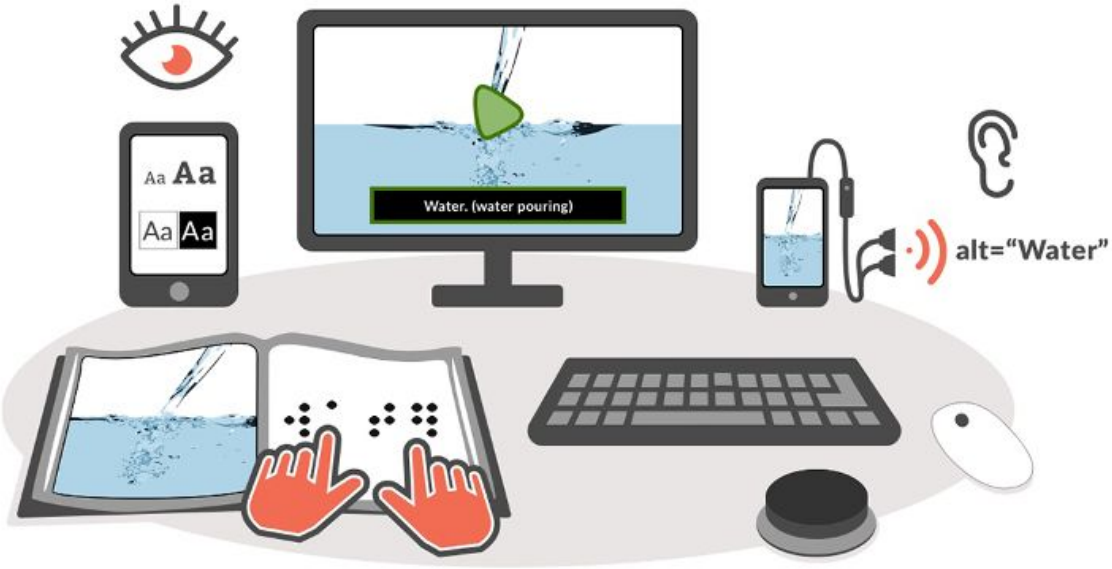
Intro to React and Firebase

Agenda

- Accessibility
- JavaScript Overview
- React
- Firebase
- Firebase Demonstration



A11y (Accessibility)



Making Apps Accessible

- Readable fonts
  - Easily legible
  - Large font size
- Images
  - Include alt text whenever possible
- Colors
  - Color contrast
  - Color is an addition, not a necessity

THIS IS NOT OKAY

This is visible

```
<img src="img_girl.jpg" alt="Girl in a jacket"
width="500" height="600">
```

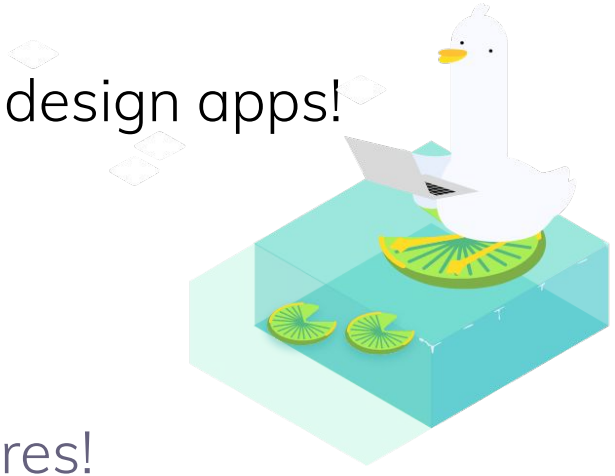
AVOID RED AND GREEN

AVOID SIMILAR COLORS

so … what should *you* do?

Start with:

- thinking about color contrast when you design apps!
- check color contrast on new projects



Integrate this into your workflow:

- Figma plugins!
- Chrome and Firefox have dev tool features!

If you're interested - Web Accessibility: Understanding Colors and Luminance (MDN) is a stellar resource!

so … what should *you* do?

Start with:

- consider when elements should be an image or an icon instead of text
- provide alt text for images, animations, videos, and other visual elements

Integrate this into your workflow:

- include screen reader checks in content/product audits
- use accessibility linting and continuous integration tools

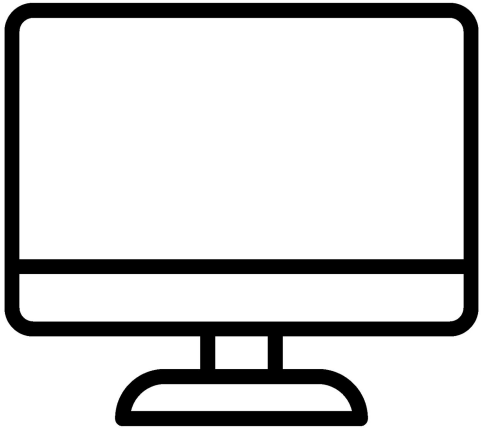
Great starting point: [How to Design Great Alt Text: An Introduction (Deque)]

a lot we didn't cover!

- use **large, legible fonts**
- offer options to **reduce motion** (autoplaying, transitions, gifs)
- offer closed captioning/subtitles (Zoom has this!)
- think about the language you're using; is it **too domain-specific?**
- if you design interfaces: **how diverse is your user testing pool?**
- If you implement interfaces: are you **encoding accessibility** in your work?

But, long story short, disability is a *huge* spectrum; nothing is comprehensive.

General Computer Science Concepts



Objects and Variables

- VARIABLES: hold values
- Can hold numbers, words, letters

```
var age = 5;
var name = "Bob";
var letter = 'b';
```

- General variables are declared using the word var followed by the variable name

```
var variable_name = value;
```

- Using variables
  - Adding

```
var age = 5;
var yearsPassed = 4;
age += yearsPassed;
//age is now equal to 5+4=9
var firstName = "Bob";
var lastName = "the Builder";
var name = firstName +
lastName;
//name = "Bob the Builder"
```

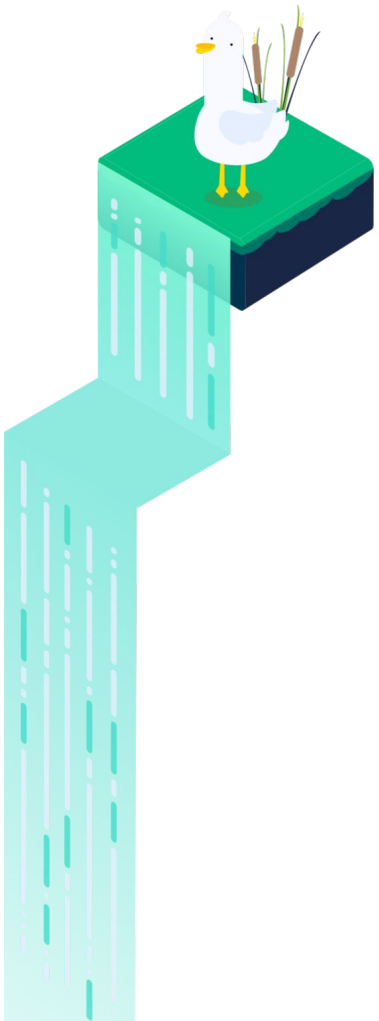
Objects and Variables

- A book has properties and type
  - Title, author, publication date, etc.
- OBJECTS in Javascript also have properties and type
  Ex: book
- We create an object using const

```
const book = {
author: "Jane Austen",
title: "Pride and Prejudice";
date: 1813 };
```

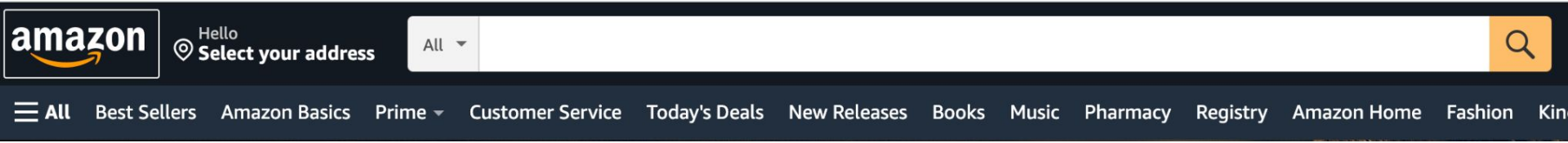
- Accessing objects: objectName.propertyName

```
book.author = "Bob";
book["title"] = "Flowers";
```



Lists

- A collection of items
  - Groceries, student names, clothes
- Helpful for menus
- Used with JavaScript “arrays” and keys
  - Keys identify what is in a list



Lists

Example of what a list looks like: taking an array and producing a list

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>

    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

```
const numbers = [1, 2, 3, 4, 5];
const root =
ReactDOM.createRoot(document.getEl
ementById('root'));

root.render(<NumberList
numbers={numbers} />);
```

If/Else Conditionals

- If/Else conditions are a way to control how your code works depending on your variables.
- Here is an example in English: if it is snowing, wear a heavy jacket. Else if it is raining, wear a rain jacket. Else, wear a t-shirt.
- Now we'll see the same concept but in JavaScript.

```
bool isSnowing = false;
bool isRaining = true;
if(isSnowing) {
        wearHeavyJacket();
}
else if(isRaining) {
        wearRainJacket();
}
else {
        wearTShirt()
}
```

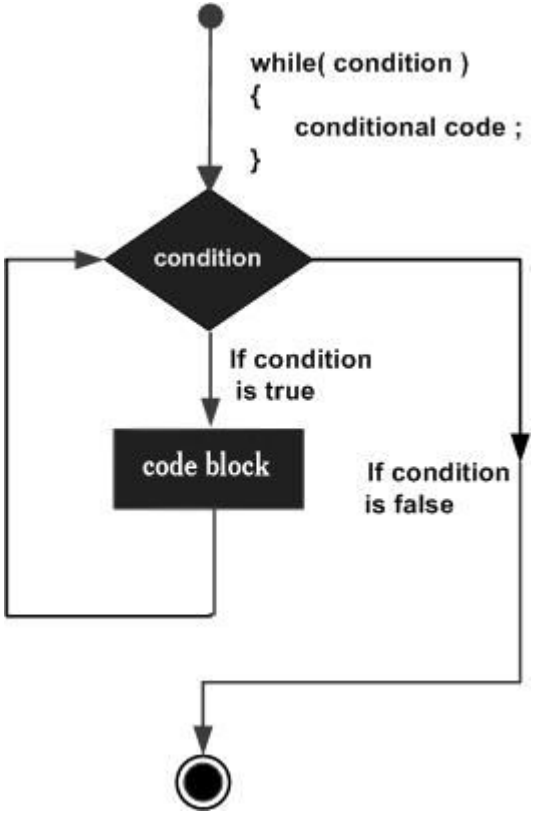
Loops

- Repeating the same code multiple times
- We control how much code repeats itself:
  - Set amount of times (I want to print the first ten items in a list)
  - Until a condition is met (Keep printing every item in a list until there are no more left)
- Two ways to repeat code in programming: while loops and for loops

While Loops

```
while (condition) {
  // code
  // so-called "loop body"
}
```

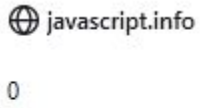
- While the condition is true, run the code
- Execution of loop body - iteration

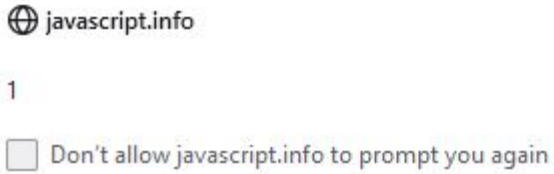


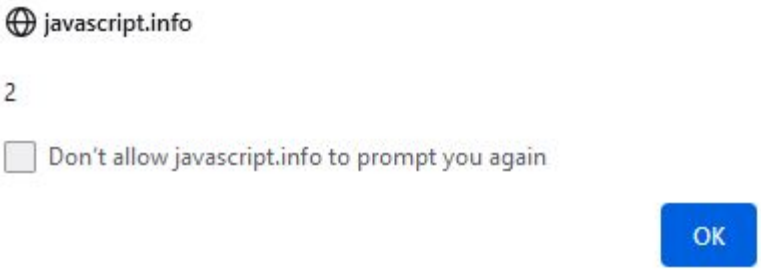
While Loops

- Example: loop iterates 3 times, displaying 0 to 2
- Note: alert sends a pop-up box with the message inside the parenthesis

```
let i = 0;
while (i < 3) { // shows 0, then 1, then 2
  alert( i );
  i++;
}
```



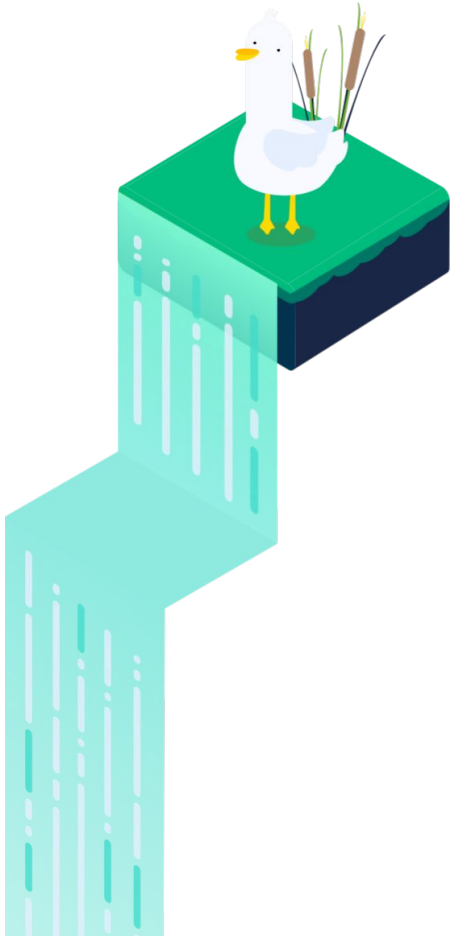




While Loops

- What happens if we remove i++?

```
let i = 0;
while (i < 3) { // shows 0, then 1, then 2
  alert( i );
  i++;
}
```



Do... While Loop

- The loop runs the code *first*, then checks the condition
- This means the code will always run <u>*at least once*</u>

```
do {
  // loop body
} while (condition);
```

```
let i = 0;
do {
  alert( i );
  i++;
} while (i < 3);
```

For Loops

- Begin: let i = 0
  - Runs once when the loop starts
- Condition: i < 3
  - Checked before every iteration, loop stops once false
- Body: alert(i)
  - Runs again and again while condition is true
- Step: i++
  - Executes after the loop body

```
for (begin; condition; step) {
  // ... loop body ...
}
```

```
for (let i = 0; i < 3; i++) { // shows 0, then 1, then 2
  alert(i);
}
```

JavaScript Functions

- For when you want to use code again and again and again
- Basis of many principles in JS
- Example:
  - Define the function LA, taking a parameter hacks
  - Returns the value of hacks + 2
  - Called with the argument 23 in place of hacks, so returns 25

```
function LA(hacks) {
  let foo = hacks + 2;
  return foo;
}

console.log(LA(23));
```

Anonymous Functions

- In JS, you can pass functions as arguments to other functions
- This means you sometimes want to make a function without a name
- Example:
  - biggerHacks: increment every element by 2
    - biggerHacks = [3,5,45,25]
  - complexHacks: replace every value with a string depending on the value
    - complexHacks = [“cool”, “cool”, “swag”, “cool”]
- Makes code much neater!

```
1   let hacks = [1, 3, 43, 23];
2
3   let biggerHacks = hacks.map((singleHack :number ) => hack + 2);
4
5   let complexHacks = hacks.map((singleHack :number ) => {
6     if (singleHack  < 42){
7       return "cool";
8     } else {
9       return "swag";
10    }
11  })
12
13  console.log(biggerHacks)
14  console.log(complexHacks)
```

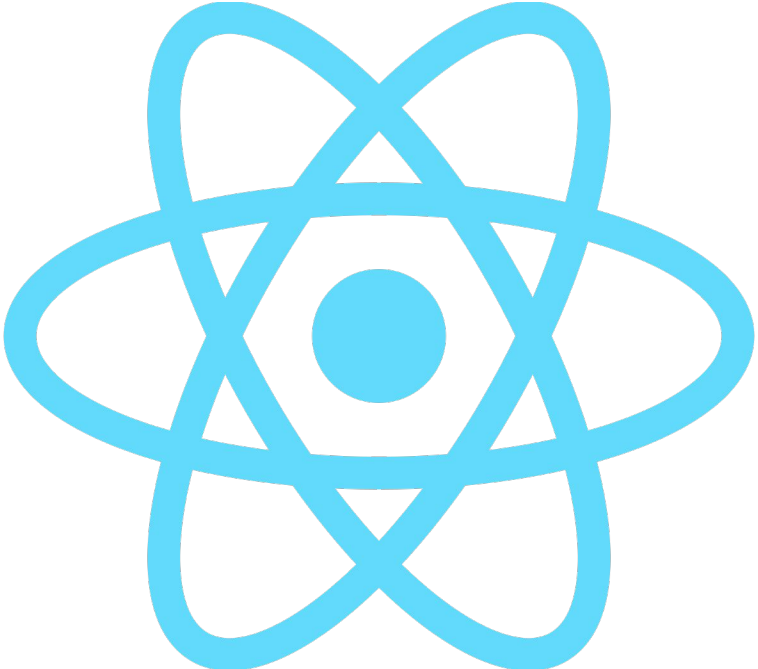
More functions

- You can also create a function like another variable
- This function returns the same value as the one we wrote before
- If functions are declared this way, they will have the scope a regular variable would

```
1  function LA(hacks) {
2    let foo = hacks + 2;
3    return foo;
4  }
5
6  console.log(LA(23));
```

```
1  const LA = (hacks) => {
2    return hacks + 2;
3  }
4
5  console.log(LA(23));
6
```

Javascript and React



Web Development Languages

- HTML
  - Language used to display content in the browser
- CSS
  - Styling (make things look cool)
- JavaScript
  - Add logic to your website

HTML

```
<h1>Hello World</h1>
```

Hello World

Opening Tag	Closing Tag	Description
		Section
<h1>	</h1>	Main heading
<h2>	</h2>	Smaller heading
		Paragraph

This is a header

This is a sub header

This is a paragraph

```
<div>
    <h1>This is a header</h1>
    <h2>This is a sub header</h2>
    <p>This is a paragraph</p>
</div>
```

Opening Tag	Closing Tag	Description
	None	Image
		Link

CSS

- Styles override each other
- Specify element you want to style and the properties of it

Syntax:

```
element {
    properties;
}
```

CSS - Common Examples

```
p {
background-color: green;
}
```

```
p {
    color: green;
}
```

```
p {
    font-family: "Times New Roman";
    font-size: 20px;
    font-weight: 200;
}
```

CSS - Common Examples

```
img {
    border: 4px solid;
    border-radius: 10px;
    border-color: green;
    box-shadow: 10px 5px 5px
black
}
```

```
img {
    border-color: #c960ff;
    padding: 50px;
    margin: 50px;
}
```

CSS

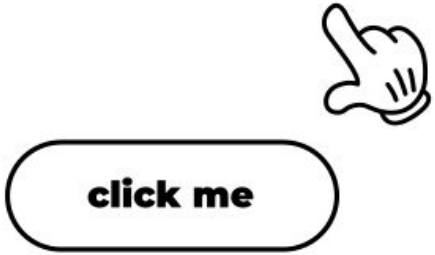
- `` `className = "className"` `` in HTML
- `` `.className {}` `` in CSS
- `` `id = "idName"` `` in HTML
- `` `#idName {}` `` in CSS
- `<p style={{ fontSize: "25px" }}>`

Javascript

- Scripting language (runs when a page loads, instead of compiling/executing)
- Programs behavior of web pages (interactivity)

What is React?

- JavaScript library for user interfaces (menus, buttons, searchbars)
- Apps! Websites! More!
- Easy, intuitive, popular, career development



Components

- Core concept of React and building UIs
- Splits websites into reusable "components", each with its own functionality

Components

```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
```

Functional Component

Components

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

Class Component

```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
```

Functional Component

JSX

- Stands for JavaScript XML, in HTML style
- JSX describes the content within the parenthesis
- Doesn't look like JavaScript but React changes it into a form understandable by the browser
- Benefits of JSX - React components within components, simplifying complex UI

```
function WelcomeMessage() {
  return <p>Welcome!</p>
}

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo"
alt="logo" />
        <p>
          Edit <code>src/App.js</code> and
save to reload.
        </p>
        <WelcomeMessage />
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  )
}
```

Exporting

- Export component in component file
  - export default at end of file to export single functional component

```
3   function Gloves() {
4       return (
5           <div>
6               <h1>These are gloves. </h1>
7           </div>
8       )
9   }
10
11  export default Gloves;
```

Importing

- Import component in file to use it
  - Import Component from ‘./ComponentFile’;

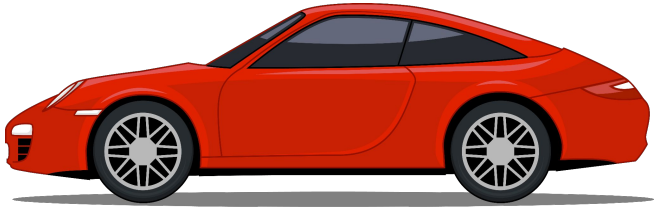
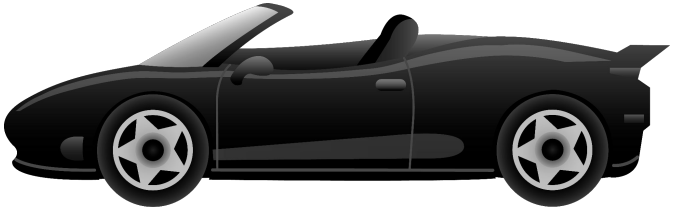
```
import Gloves from "./components/Gloves";
```

```
<Gloves />
```

Customizability

- Our components always render the same thing

- How can we customize our components?


Props

- Also known as “properties”
- Let you pass data into existing HTML tags and your own components
- Think of them like function arguments
- className, src, alt, width, and height are props you pass to an <img>

```
function Avatar() {
  return (
    <img
      className="avatar"
      src="https://i.imgur.com/1bX5QH6.jpg"
      alt="Lin Lanying"
      width={100}
      height={100}
    />
  );
}
```

Output:

Passing in Props

- Can be used for user-defined components
- Passing person (object) and size (number) props into Avatar component
- <Component propName = propData />
- { } lets you put Javascript code in HTML (JSX!)
- Another { } to denote an object in JS
- export default allows you to import this function in other files

```
export default function Profile() {
  return (
    <Avatar
      person={{ name: 'Lin Lanying', imageId: '1bX5QH6' }}
      size={100}
    />
  );
}
```

Reading in Props

- You can then use these props inside your Avatar component like variables
- After function, list the names of each prop separated by commas inside {{ }} (object destructuring)

```
export default function Profile() {
  return (
    <Avatar
      person={{ name: 'Lin Lanying', imageId: '1bX5QH6' }}
      size={100}
    />
  );
}
```

```
function Avatar({ person, size }) {
  // person and size are available here
}
```

Props

- Now we rewrote Avatar to use props instead of hardcoding values
- The values from the props are passed into from the Profile function
- Accessing props: {propName}
- Code reusability, customizability

```
export default function Profile() {
  return (
    <div>
      <Avatar
        size={100}
        person={{
          name: 'Katsuko Saruhashi',
          imageId: 'YfeOqp2'
        }}
      />
      <Avatar
        size={80}
        person={{
          name: 'Aklilu Lemma',
          imageId: 'OKS67lh'
        }}
      />
      <Avatar
        size={50}
        person={{
          name: 'Lin Lanying',
          imageId: '1bX5QH6'
        }}
      />
    </div>
  );
}
```

```
function Avatar({ person, size }) {
  return (
    <img
      className="avatar"
      src={getImageUrl(person)}
      alt={person.name}
      width={size}
      height={size}
    />
  );
}
```

Output:

Parent-Child Components

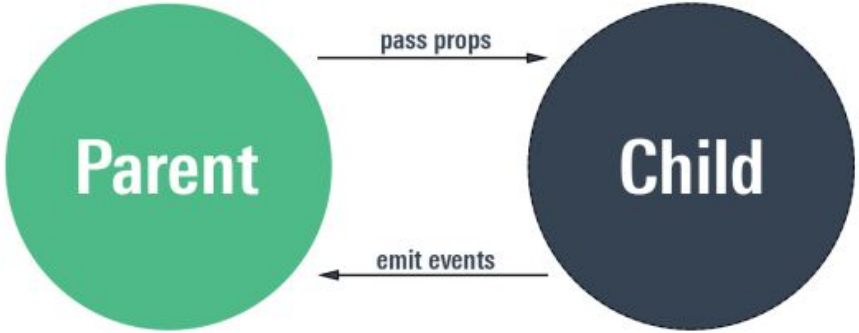
- One-way transfer of data, only parent component to child component
- Parent: Component you're currently working inside
- Child: Component you are using (rendering) within parent
- Props: read-only and cannot be modified

Parent:

```
function App() {
  return (
```

Child:

```
      <Gloves />

    </div>
  );
}

export default App;
```



State?

- Sometimes you need to to display a variable on the page
- Let's take a look at an example:

```
let duckClicks = 0;
return (
  <div>
    <h1>You have clicked the duck {duckClicks} times</h1>
    <img
      src = {'https://lahacks.com/static/hero/bottom_duck.svg'}
      onClick = {() => duckClicks = duckClicks + 1}
      style = {{width: '25%', height: 'auto'}}
    />
  </div>
);
}

export default App;
```

- It looks like it should work, but when we click the duck, nothing changes!
- Why? We didn't use state!

React State

- State is essentially some data that belongs to the component that can be updated when necessary.
- The key is that React re-renders that component for us when updated!
- To use, we need an import statement:

```
import React, { useState } from "react";
```

State.

- If we want the page to refresh when the user clicks, we'll need to use state!

```
const [duckClicks, setDuckClicks] = useState(0);
```

- This creates a variable duckClicks(initialized to 0) and a function to set it that will set duckClicks to a different value and refresh the page
- State variables can be any value, including arrays and more complex objects
- State changed functions can also be passed as arguments, so you can pass state between components

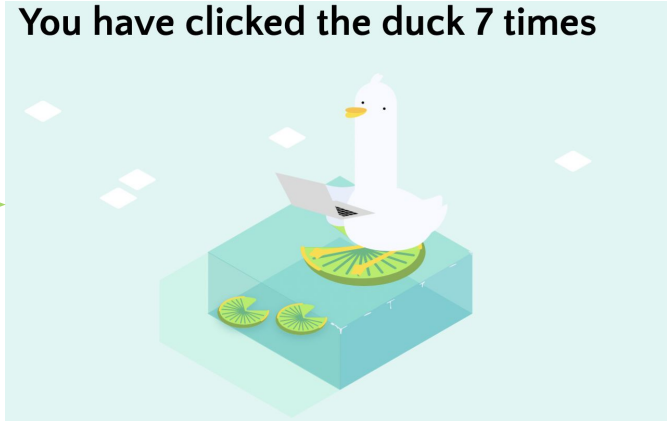
Danger: This means that when you change state, it will refresh the page immediately! So be careful not to create infinite loops!

State!

- Here, we use state to store duckClicks instead!

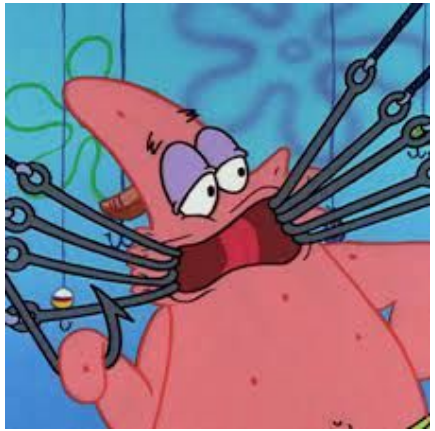
```
function App(): JSX.Element {
  const [duckClicks, setDuckClicks] = useState( initialState: 0);
  return (
    <div style={{background:"#ddf6f3"}}>
      <h1>You have clicked the duck {duckClicks} times</h1>
      <img
        src = {'https://lahacks.com/static/hero/bottom_duck.svg'}
        onClick = {() => setDuckClicks( value: duckClicks + 1)}
        style = {{width: '25%', height: 'auto'}}
      />
    </div>
  );
}

export default App;
```

- Now, the page resets every time setDuckClicks() is called, so the action is shown on the page
- So it works!



React Hooks

- What we just used to manage state is an example of a Hook
- Hooks are special functions that let you easily change information in components!
  - Basically, they help update the appearance of your site



React Hooks

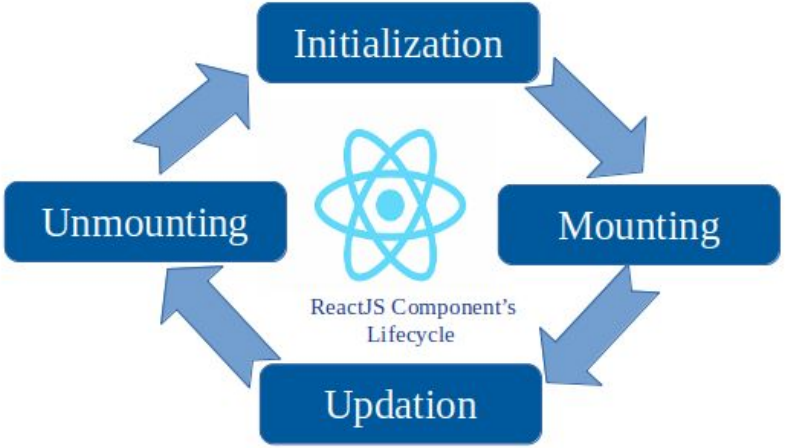
- Two most common hooks:
- useState - way to change certain variables due to external events on your site
- useEffect - monitor the state of something and run code whenever it changes

useEffect

```
useEffect(() => {
  console.log("hi");
}, []);
```

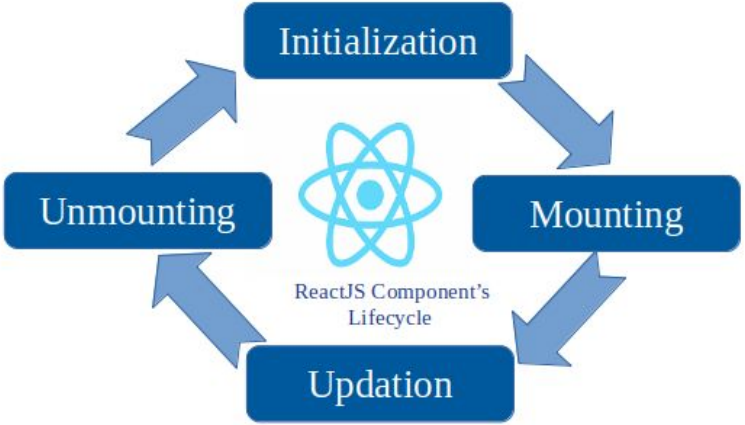
useEffect

- useEffect can run code in these scenarios:
  - When a component is mounted
  - When a component is updated
  - When a component is unmounted



useEffect

- useEffect: good for running code without user interaction (example: pulling data from Web APIs)
- useState: good for running code based on user interaction (example: clicking a button)



useEffect

```
import React, { useEffect } from 'react';
```

useEffect

```
useEffect(() => {
  console.log("hi");
}, []);
```

- Anonymous arrow function (Javascript)
- Callback: code we want to run (console.log(“hi”))
- Dependencies: how often we want to run the callback
  - [] => only runs callback when component mounts

useEffect

```
function App() {
    const [text, setText] = useState("Hi kitten ;)");

    useEffect(() => {
        console.log("triggered!");
    }, [text]);

    return (...);
}
```

This runs every time `text` changes value

Setup

Get Node

- https://nodejs.org/en/download/
- Download for your respective laptop
- **Next -> Next -> npm package manager -> Next -> Install**

1. **Open Terminal**
2. **Create a new project:**
   a. mkdir ReactProjects
3. **Enter project directory**
   a. cd ReactProjects
4. **Install React**
   a. npx create-react-app my-app
5. **Change Directories**
   a. cd my-app
6. **Start React App**
   a. npm start

React Demo

Image size is 300px by 300px

You have clicked the duck 0 times.

Reset

Image size is 400px by 400px

You have clicked the duck 10 times.

Reset

React Demo Code

https://github.com/nwang03/lahacks23demo

Databases

Why Use Databases?

- Databases provides a central location for storing user information without compromising local storage
- Allows
  - Authentication
  - File/image upload

Key Ideas

Stores Various Data Types

- ints/floats
- strings
- booleans
- JSON objects

Organized Into

- Tables
  - Columns
  - Rows

Uses

- Key Value Pairs
- Ex:
  - Key could be a user ID
  - Value could be the user's data.

Key = id

Value = all other columns

Table: customers

customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Taylor	806-749-2958	UAE

Multiple Databases

Table: orders

order_id	product	total	customer_id
1	Paper	500	5
2	Pen	10	2
3	Marker	120	3
4	Books	1000	1
5	Erasers	20	4

Table: customers

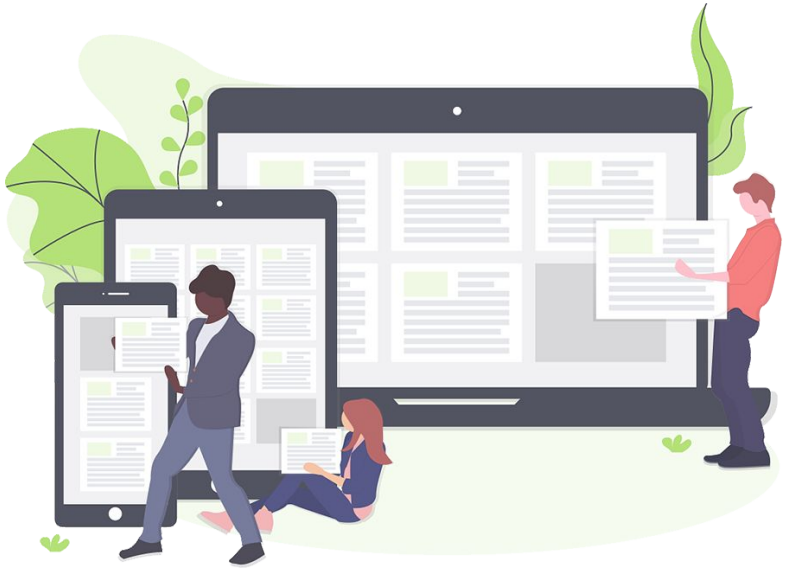
customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Doe	806-749-2958	UAE

Firebase

What is Firebase

A “comprehensive app development platform” which helps manage your database

- Authentication
- Databases
- File and Image Storage
- Real Time Database Updates
- Firebase Queries
- Hosting an Application



Storing data

```javascript
import{getDatabase, ref, set} from "firebase/database"

function hotelBookings(bookingNum, hotelName, last_name, number, userID) {
  const d = getDatabase();
  const reference = ref(d, 'hotel bookings/' + bookingNum);

  set(reference, {
    hotel: hotelName,
    last_name: last_name,
    phone_number: number
    userId: userID
  });
}

hotelBookings(num, "Marina Bay Sand", "Tan", 20160102, ID);
```

```
project-5939797027399584108
  hotel bookings + ×
    -K2ib4H77rj0LYewF7dP
      hotel: "Marina Bay Sand"
      last name: "Tan"
      number: "20160102"
      userID: "Wwevm5nkpyfAd8dZPMVPJiNqVdQ2'
    -K2ib5JHRbbL0NrztUfO
      hotel: "Good Wood Hotel"
      last name: "Ong"
      number: "20160129"
      userID: "o11Q4w81dxf9iSbA6SzrfEwkAYs1'
    -K2ib62mjHh34CAUbide
      hotel: "Raffles Hotel"
      last name: "Lim"
      number: "20160120"
      userID: "GEmgtbLrs4c4puQWN5i1SulsLjj2'
```

Storing images

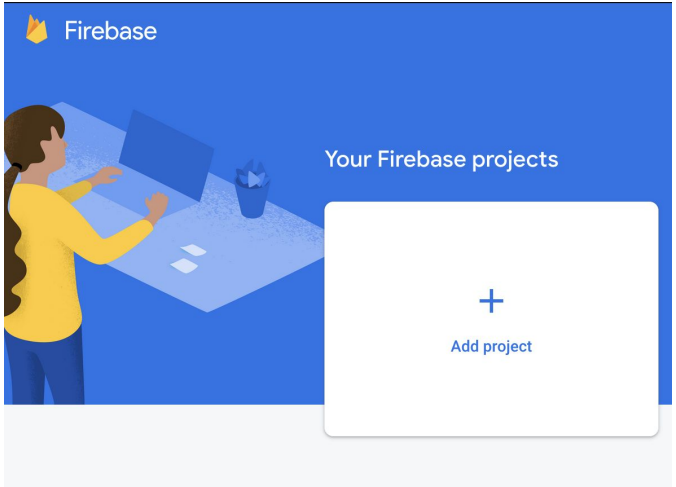
```
import{getDatabase, ref, set} from "firebase/database"

function hotelBookings(bookingNum, hotelName, last_name, number, userID, imageURL) {
  const d = getDatabase();
  const reference = ref(d, 'hotel bookings/' + bookingNum);

  set(reference, {
    hotel: hotelName,
    last_name: last_name,
    phone_number: number
    userId: userID
    image: imageURL
  });
}

hotelBookings(num, "Marina Bay Sand", "Tan", 20160102, ID, "myimageurl")
```

gs://cropped-images.appspot.com > images

Name	Size	Type
1d7ce39bef9b21a0c9f6dd30f6e43e44.jpg	28.02 KB	image/jpeg
1d7ce39bef9b21a0c9f6dd30f6e43e44Cropp...	13.11 KB	image/jpeg
2d667c1606e23cabe4c7ccd442c2aa2a.jpg	59.71 KB	image/jpeg
2d667c1606e23cabe4c7ccd442c2aa2aCrop...	30.49 KB	image/jpeg
64fe9f9e62e628ef4a949339d9ac63de.jpg	111.05 KB	image/jpeg
64fe9f9e62e628ef4a949339d9ac63deCropp...	24.84 KB	image/jpeg
e39e37a9ad04ffb0a653f966d385a65fCropp...	33.16 KB	image/jpeg
f8c5c821649aa1a9a49f9e6825b1dd85.jpg	41.78 KB	image/jpeg
f8c5c821649aa1a9a49f9e6825b1dd85Crop...	22.52 KB	image/jpeg
Tang.png	118.05 KB	image/png

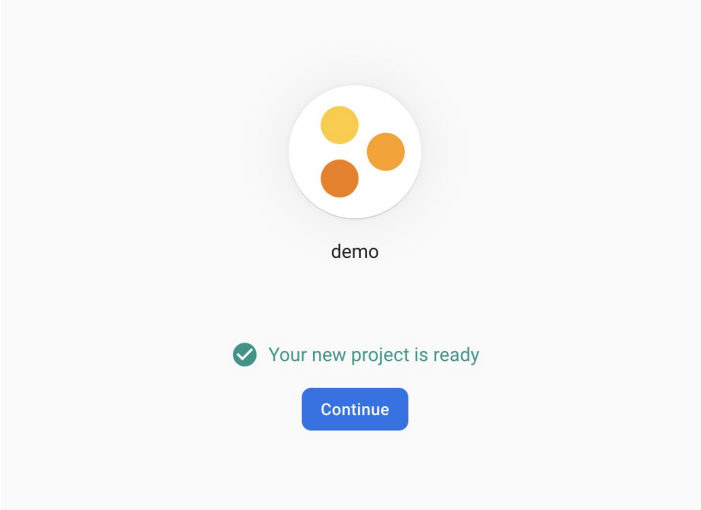
Setting Up

- Go to https://console.firebase.google.com/u/0/
- And click the “Add project”button

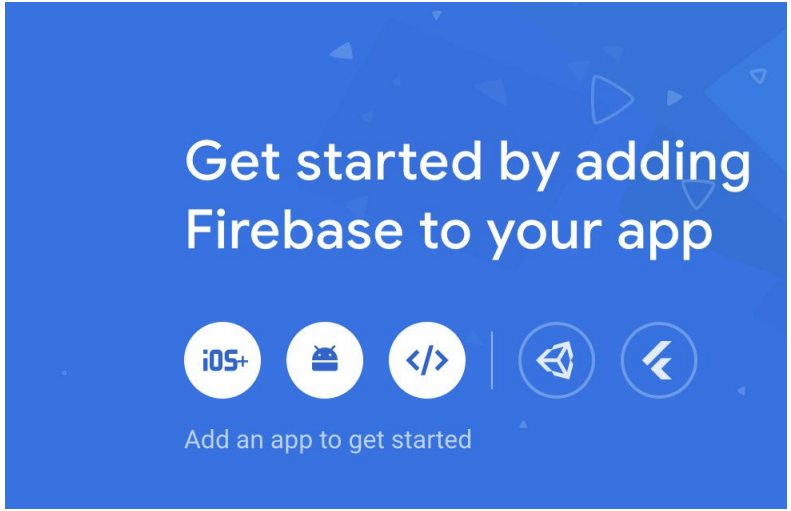


Setting Up (Cont.)

- Come up with a name for your project
- Enable Google Analytics (default)
- And click the create your project button



Linking Firebase to your project



- Choose a type of app to link Firebase to
  - If you don't see this page, click settings → project settings, and scroll down to "your apps".
- Give your app a name, then Firebase will give you some options to add their sdk to your project

Linking Firebase to your project

- If you are creating a web-app, Firebase gives you the option to use npm, or use <script> tags

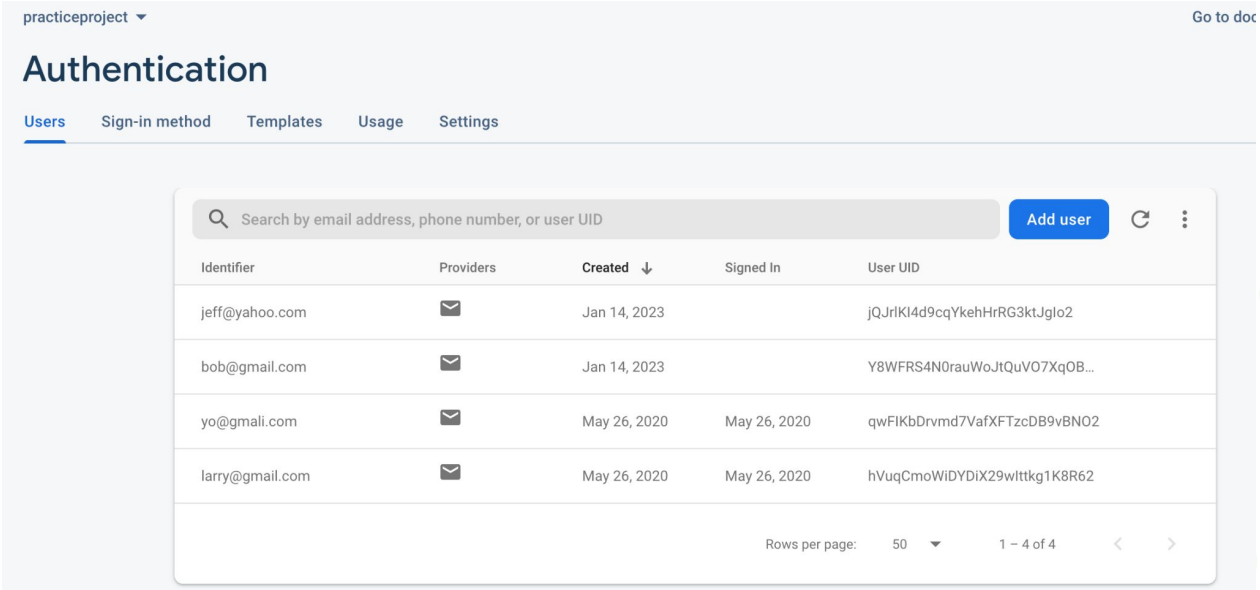
- Android and IOS apps have a few additional steps that are shown on the Firebase website

```
const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
const provider = new GoogleAuthProvider();
export const signInWithGoogle = () => {
 signInWithPopup(auth, provider)
   .then((result) => {
     const name = result.user.displayName;
     const email = result.user.email;
     const profilePic = result.user.photoURL;
     localStorage.setItem("name", name);
     localStorage.setItem("email", email);
     localStorage.setItem("profilePic", profilePic);
     refreshPage();
     // console.log(name);
   })
   .catch((error) => {
     console.log(error);
   });
};
```

Logon Example

Creating a form to allow new users to register with an email and password

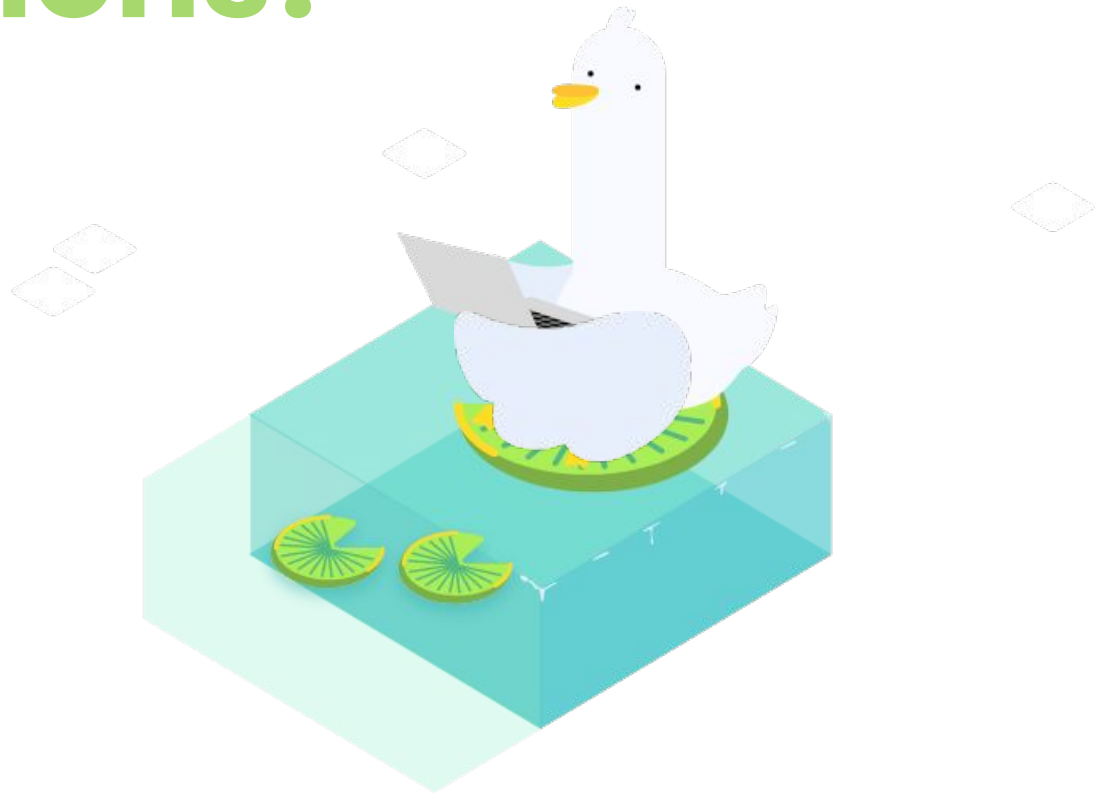
Managing Users



The registered users can be viewed from the firebase console under authentication

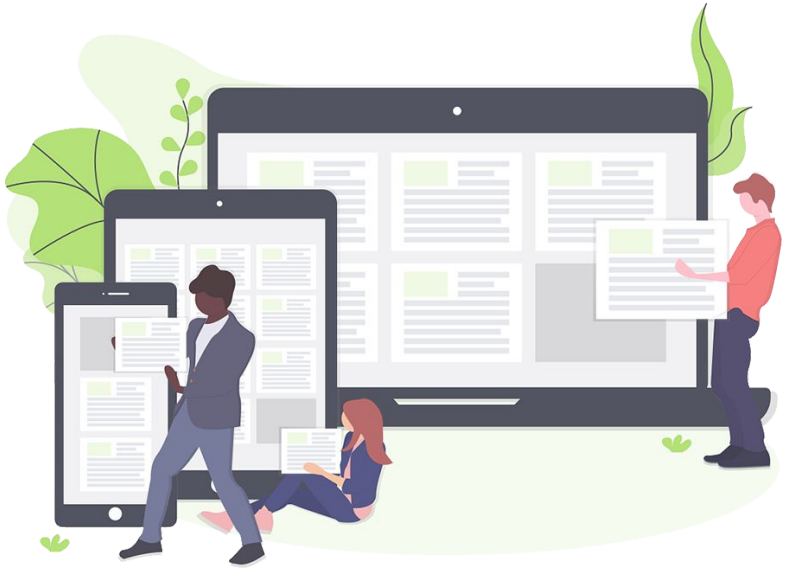
Go to → https://firebase.google.com/docs/auth for questions with auth

Questions?



Recap

- How to promote accessibility in digital content
- Basic JavaScript overview
- React concepts
- Firebase overview
- Firebase walkthrough



Happy coding!