



REACT FIREBASE

Lookup.KeyValue
f.constant(['en-US'])
=tf.constant([0])
.lookupStaticValue
Avalon AI
buckets=5)



AUGISTA EKSOFON PRADANA

SOFTWARE ENGINEERING, DEPARTMENT OF INFORMATICS ITS

Experience

- Expert Staff IT Dev Mabacup 2024
- Expert Staff IT Dev Schematics ITS 2025
- Chief Technology Officer of PT. CoNation
- Frontend Engineer at JagoTeknik (Education Startup)
- Backend Engineer at IMAC 2024

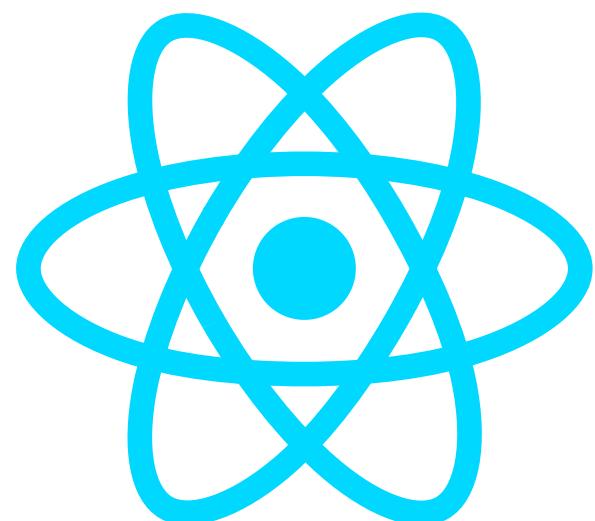
Organization

- Head Bureau C of Internal Affairs HMTC ITS 2025
- Project & Journal Staff IEEE ITS Student Branch 2024
- Project Analyst 180DC Consulting ITS





WHAT IS REACT?



React

- React is a JavaScript library for building user interfaces.
- Developed by Facebook (now Meta).
- Purpose: Create fast, scalable, and simple UI for web applications.
- React is Component-based architecture.

WHY REACT MOST USED FOR WEB DEV?

- **Reusable Components:** Build once, use everywhere.
- **Virtual DOM:** Faster rendering by updating only changed elements.
- **Unidirectional Data Flow:** Makes data handling easier to manage.
- **Strong Ecosystem:** Rich community and tools.





Google Developer Groups



REACT COMPONENT

```
import React from 'react';

function WelcomeMessage() {
  return <h1>Welcome to React!</h1>;
}

export default WelcomeMessage;
```

Components: The building blocks of a React application.



JSX

What is JSX?: A syntax extension that allows mixing HTML and JavaScript.

Benefit JSX :

Readable and declarative UI code.

```
const element = <h1>Hello, World!</h1>;
```



```
dren: [  
  icon(icon, color: color  
  container(  
    margin: const EdgeIns  
    child: Text(  
      label,  
      style: TextS
```

STATE MANAGEMENT

- State: Used to store data that can change over time.
- Function : State updates trigger a re-render of the component.



```
import React, { useState } from "react";  
  
const Counter: React.FC = () => {  
  const [count, setCount] = useState<number>(0);  
  
  return (  
    <div>  
      <p>Count: {count}</p>  
      <button onClick={() => setCount(count + 1)}>Increment</button>  
    </div>  
  );  
};  
  
export default Counter;
```

useState manages the count state.
Clicking the button increments the count.



PROP & EVENT HANDLING

PROP

Props (Properties)

Props: Read-only data passed from parent to child components.

Passing data to child components using props:

```
function Greeting(props: { name: string }) {
  return <h1>Hello, {props.name}!</h1>;
}

export default function App() {
  return <Greeting name="Steve" />;
}
```



EVENT HANDLING

```
function ClickButton() {
  const handleClick = () => {
    alert('Button clicked!');
  };

  return <button onClick={handleClick}>Click Me</button>;
}

export default ClickButton;
```

onClick is used to attach the handleClick function to the button.



HOOK

- Hooks: Functions that allow functional components to manage state and lifecycle.
- Common hooks:
- useState: Manage state.
- useEffect: Handle side effects like data fetching.
- useContext: Manage global state.



```
keyValue  
int(['em  
stant([0  
StaticV  
=5)
```





useEffect Hook

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

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds((prev) => prev + 1);
    }, 1000);
    return () => clearInterval(interval);
  }, []);

  return <p>Timer: {seconds} seconds</p>;
}

export default Timer;
```

useEffect sets up an interval and cleans it up when the component unmounts





useContext Hook

```
import { createContext, useContext, useState } from "react";

const ThemeContext = createContext();

const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState("light");

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

export { ThemeProvider, ThemeContext };
```

- **createContext()** creates a global context.
- **useContext(ThemeContext)** is used in components to get the value of the context.
- Allows us to change themes without having to manually pass props to each component.



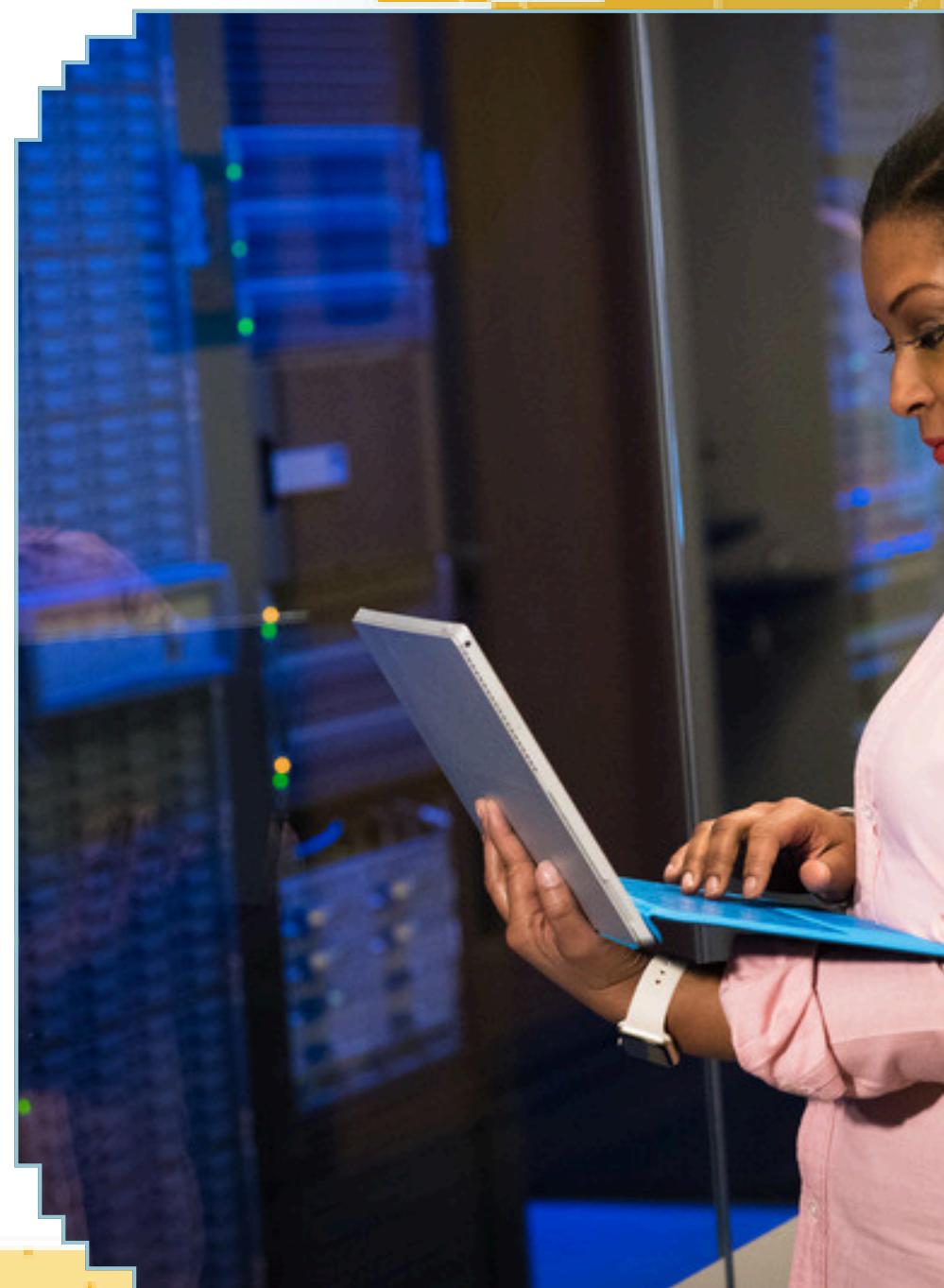
useRef Hook

useRef is a React Hook used to:

- Save a reference to a DOM element without causing a re-render.
- Store a value between re-renders without causing a state change.

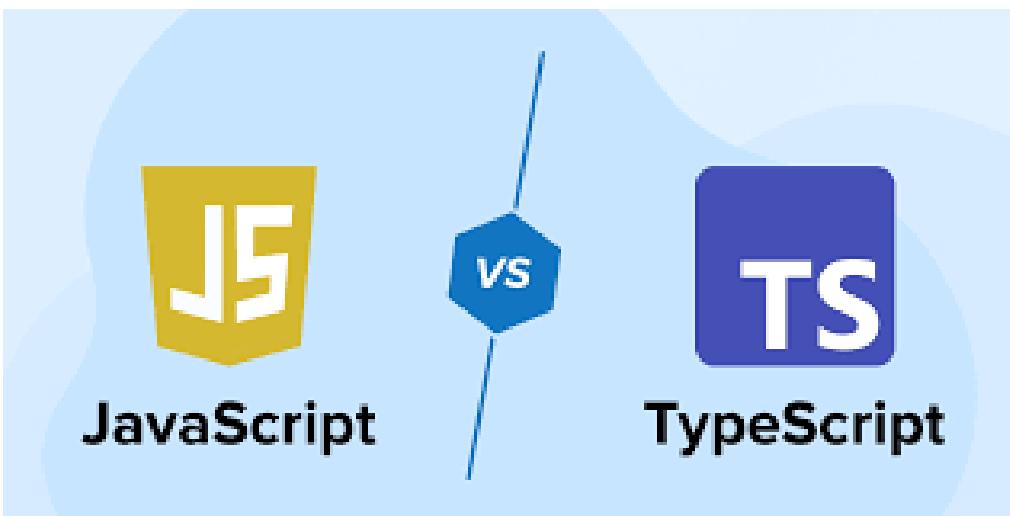
```
const scroll = useRef<HTMLDivElement | null>(null);
const chatContainerRef = useRef<HTMLDivElement | null>(null);
```

- ◆ **scroll** → Used to handle automatic scroll down when there is a new message.
- ◆ **chatContainerRef** → Used to store the chat container reference so that it can be accessed directly.



WHAT DIFFERENT BETWEEN .JS VS .TS?

Perbedaan	JSX (JavaScript XML)	TSX (TypeScript XML)
Tipe Data	Tidak memiliki tipe data bawaan	Memiliki static typing
Error Handling	Error hanya muncul saat runtime	Error terdeteksi saat compile
Keamanan	Kurang aman karena tidak ada validasi tipe	Lebih aman karena memiliki type checking





WHAT DIFFERENT BETWEEN .JS VS .TS?

```
const Greeting = ({ name }) => {
  return <h1>Hello, {name}!</h1>;
};
```

There's No type Checking for **name**

```
interface Props {
  name: string;
}

const Greeting: React.FC<Props> = ({ name }) => {
  return <h1>Hello, {name}!</h1>;
};
```

Props prohibits **name** must **string** type



WHAT IS TYPESCRIPT?



TS

A large, bold, white 'TS' logo is centered on a solid blue square background.

```
Lookup.KeyValue  
f.constant(['e'])  
=tf.constant([0])  
.lookup.StaticV  
_buckets=5)
```

A black rectangular box containing several lines of code related to TensorFlow.js, specifically showing how to use the `lookup` and `buckets` methods.



- A superset of JavaScript with static typing.
- Developed by Microsoft to improve scalability and maintainability.
- Compiled to JavaScript to run in a browser.

TypeScript Advantages:

- Static Typing - Prevent errors from the start with strict data types.
- Interface & Generics - The code structure is neater and reusable.
- Support for OOP - Can use classes, inheritance, encapsulation.





REACT + TYPESCRIPT

- React is still declarative & component-based, but more strict and safe.
- Uses interfaces to define props and state.
- Can use React Hooks with clear typing.

```
import React from "react";

interface Props {
  name: string;
  age: number;
}

const UserCard: React.FC<Props> = ({ name, age }) => {
  return (
    <div className="p-4 border rounded-lg">
      <h2>{name}</h2>
      <p>Age: {age}</p>
    </div>
  );
};

export default UserCard;
```



TYPESCRIPT: FUNCTION, INTERFACE

1. FUNCTIONS WITH PARAMETER AND RETURN TYPES

IN TYPESCRIPT, WE CAN SPECIFY DATA TYPES FOR PARAMETERS AND RETURN VALUES TO MAKE THE CODE SAFER.

```
interface User {
  name: string;
  age: number;
}

function getUserInfo(user: User): string {
  return `${user.name} is ${user.age} years old.`;
}

const user: User = { name: "Augista", age: 20 };
console.log(getUserInfo(user)); // Outputnya: Augista is 20 years old.
```

```
Lookup.KeyValue
f.constant(['e'])
=tf.constant([0])
.lookup.StaticV
_buckets=5)
```





TYPESCRIPT: CLASS ACCESS IDENTIFIER & GENERIC

TYPESCRIPT SUPPORTS THE CONCEPT OF CLASS-BASED PROGRAMMING WITH ACCESS MODIFIERS SUCH AS PUBLIC, PRIVATE, AND PROTECTED.

```
class Person {  
    public name: string;  
    private age: number;  
    protected job: string;  
  
    constructor(name: string, age: number, job: string) {  
        this.name = name;  
        this.age = age;  
        this.job = job;  
    }  
  
    public introduce(): string {  
        return `Hi, I'm ${this.name}, and I work as a ${this.job}.`;  
    }  
}  
  
const person = new Person("Augista", 20, "Software Engineer");  
console.log(person.introduce());
```



TYPESCRIPT: CLASS ACCESS IDENTIFIER & GENERIC

```
function identity<T>(value: T): T {  
  return value;  
}  
  
console.log(identity<string>("Hello"));  
console.log(identity<number>(123));
```

GENERICS ALLOW US TO CREATE REUSABLE CODE FOR VARIOUS DATA TYPES.

<T> IS A VARIABLE DATA TYPE PARAMETER

- ◆ IDENTITY<STRING>("HELLO") → CALLING A FUNCTION WITH A STRING



~~NEXT~~.JS

```
Lookup.KeyValue  
f.constant(['e'  
=tf.constant([0  
.lookup.StaticV  
_buckets=5)
```



WHAT IS NEXT JS?

Next JS is React framework for building server-side rendering (SSR) and static site generation (SSG) applications.

Supports automatic routing, API Routes, and performance optimization.

Built-in support for TypeScript!



```
Lookup.KeyValue  
f.constant(['e  
=tf.constant([  
.lookup.StaticV  
_buckets=5)
```



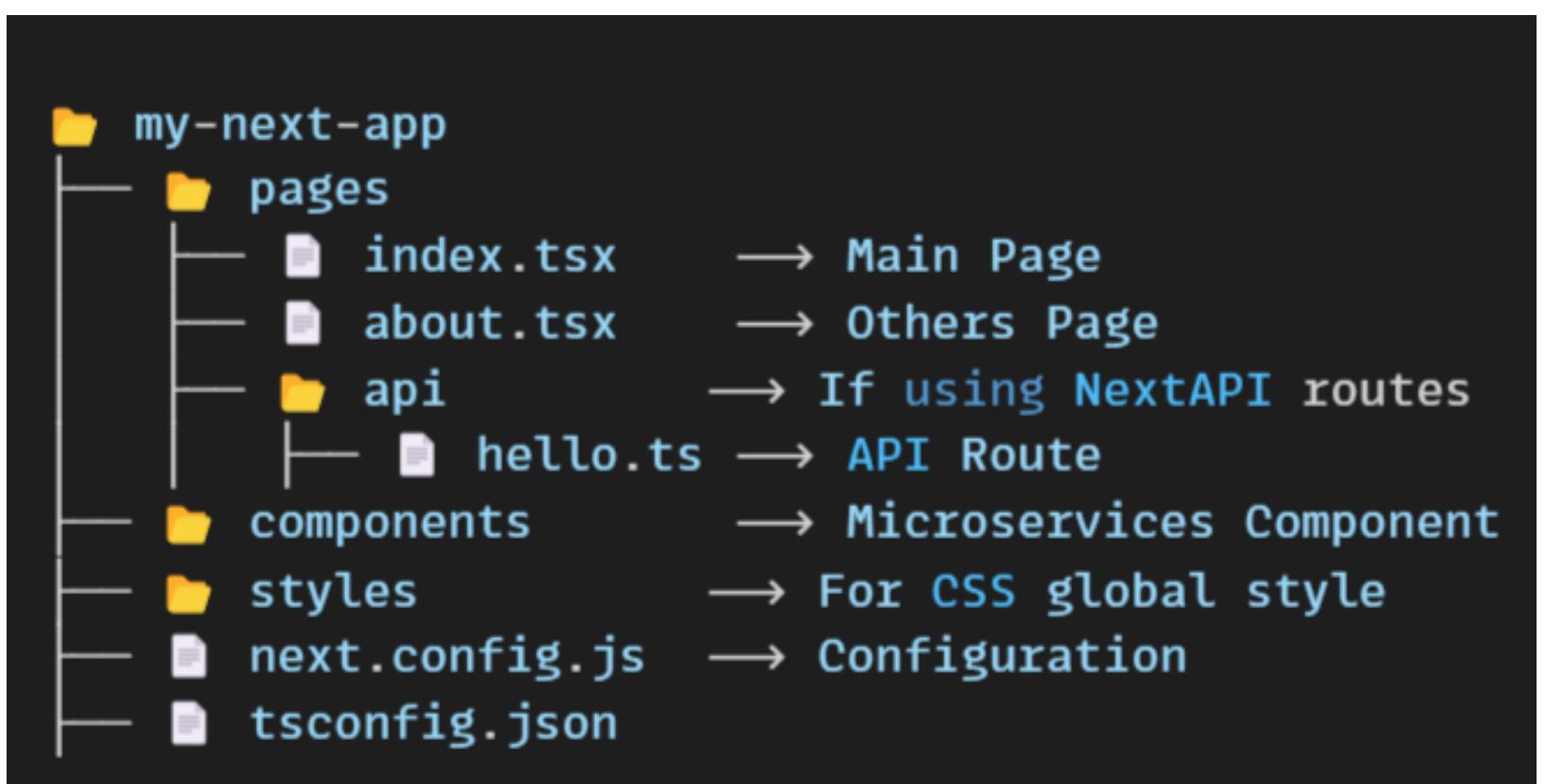
WHY WE MUST TRY NEXT JS?

- **Server-Side Rendering (SSR)** → Render pages on the server before sending them to the client.
- **Static Site Generation (SSG)** → Generate HTML at build time for faster performance.
- **API Routes** → Create backend APIs directly within your Next.js project.
- **Image Optimization** → Use next/image to load more optimized images.
- **Internationalization (i18n)** → Built-in multi-language support.

```
Lookup.KeyValue
f.constant(['en'])
=tf.constant([0])
.lookup.StaticView
buckets=5)
```



FOLDER STRUCTURE IN NEXT JS



```
Lookup.KeyValue
f.constant(['em
=tf.constant([G
.lookup.StaticV
buckets=5)
```



AUTOMATIC ROUTING

- **Next.js uses a file-based routing system, meaning every file in the pages/ folder automatically becomes a route.**
- **No need for React Router → No additional configuration required.**
- **Supports Nested Routes → Can create folders in pages/ for nested routes.**

```
import { useRouter } from "next/router";
```



Google Developer Groups

PRE-RENDERING



Next.js has two types of pre-rendering to improve performance and SEO:

Rendering When Does Rendering Happen?

- **Static Site Generation (SSG) When build time is Fast, ideal for static content**
- **Server-Side Rendering (SSR) When requesting Data is always up-to-date**

```
import { GetStaticProps } from "next";

export default function Home({ message }: { message: string }) {
  return <h1>{message}</h1>;
}

export const getStaticProps: GetStaticProps = async () => {
  return {
    props: { message: "Hello from SSG!" },
  };
};
```

```
import { GetServerSideProps } from "next";

export default function Home({ time }: { time: string }) {
  return <h1>Current Time: {time}</h1>;
}

export const getServerSideProps: GetServerSideProps = async () => {
  return {
    props: { time: new Date().toISOString() },
  };
};
```

IMAGE OPTIMIZATION



- **Automatic lazy loading**
- **Using modern formats (WebP, AVIF) for better performance**
- **Optimizing images based on screen size**

```
import Image from "next/image";
```

```
Lookup.KeyValue  
f.constant(['e  
=tf.constant([  
.lookup.StaticV  
_buckets=5)
```



MIDDLEWARE NEXTAUTH.JS (OTENTIKASI)

```
import NextAuth from "next-auth";
import GoogleProvider from "next-auth/providers/google";

export default NextAuth({
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID!,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET!,
    }),
  ],
});
```





Firebase

Using React with Firebase config to
connect realtime database and server

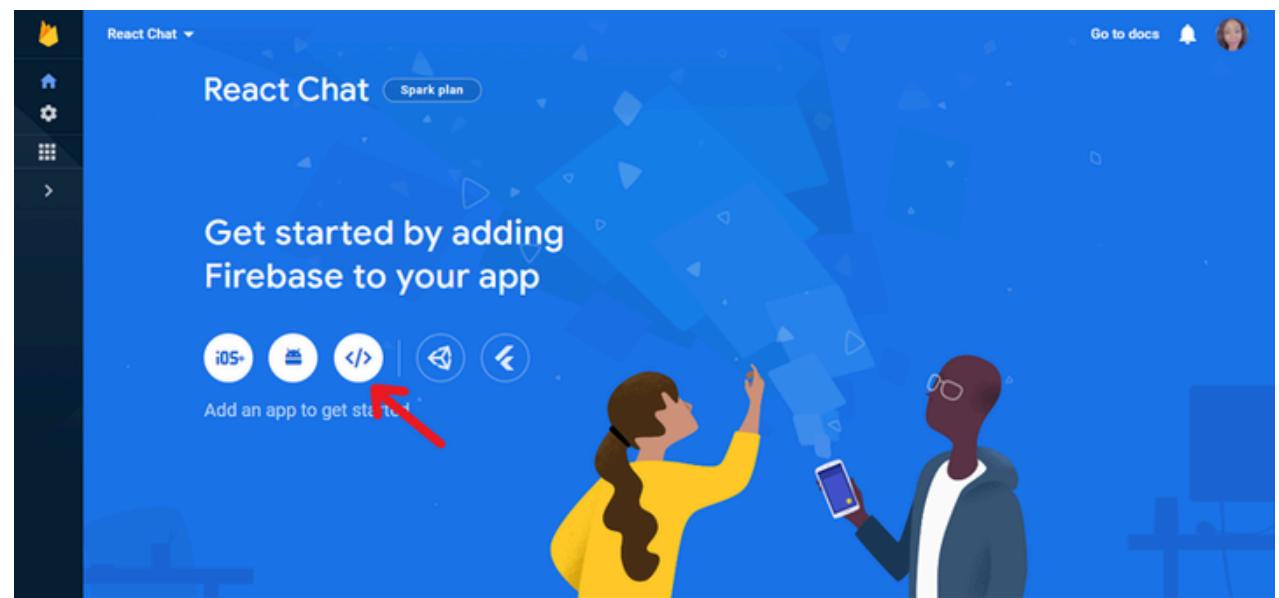


The image shows two screenshots side-by-side. On the left is the homepage of Google Developer Groups, featuring the Google logo and navigation links for Products, Solutions, Pricing, More, Search, English, Go to console, and a user profile icon. A banner at the top promotes the Firebase Summit. Below it, a large blue section says "Make your app the best it can be" with a subtext about Firebase being an app development platform backed by Google. It includes "Get started", "Try demo", and "Watch video" buttons. On the right is a screenshot of the "Create a project" step 1 of 3 page. It asks for a project name, with "React-chat" entered in the input field. It also shows a preview of the project name and a "Continue" button.

FIREBASE SETUP

isi **Create a project** form by providing a project name.

created, click on Continue.



x Add Firebase to your web app

1 Register app

App nickname

Also set up Firebase Hosting for this app. [Learn more](#)

Hosting can also be set up later. There is no cost to get started anytime.

[Register app](#)

2 Add Firebase SDK

FIREBASE SETUP

Fill in the title of the application you want to add to your Firestore



FIREBASE SETUP

2 Add Firebase SDK

Use npm [?](#) Use a <script> tag [?](#)

If you're already using [npm](#) and a module bundler such as [webpack](#) or [Rollup](#), you can run the following command to install the latest SDK:

```
$ npm install firebase
```

Then, initialize Firebase and begin using the SDKs for the products you'd like to use.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyA[REDACTED]",
  authDomain: "myapp-85d4f.firebaseapp.com",
  projectId: "myapp-85d4f",
  storageBucket: "myapp-85d4f.appspot.com",
  messagingSenderId: "9[REDACTED]7",
  appId: "1:9[REDACTED]:web:1624544444444"
};

// Initialize Firebase
```

Example after creating you will be showed Firebase SDK, you just need copy and paste to your Firebase Config (Firebase.ts)



FIREBASE CONFIG

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "API",
  authDomain: "API",
  projectId: "API",
  storageBucket: "react-chat-4c0a6.firebaseiostorage.app",
  messagingSenderId: "API",
  appId: "tesd",
  measurementId: "G-ECGGY3Y4NQ"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export const db = getFirestore(app);
```

Code Block: `firebase.ts`

Initializes Firebase using `initializeApp`.

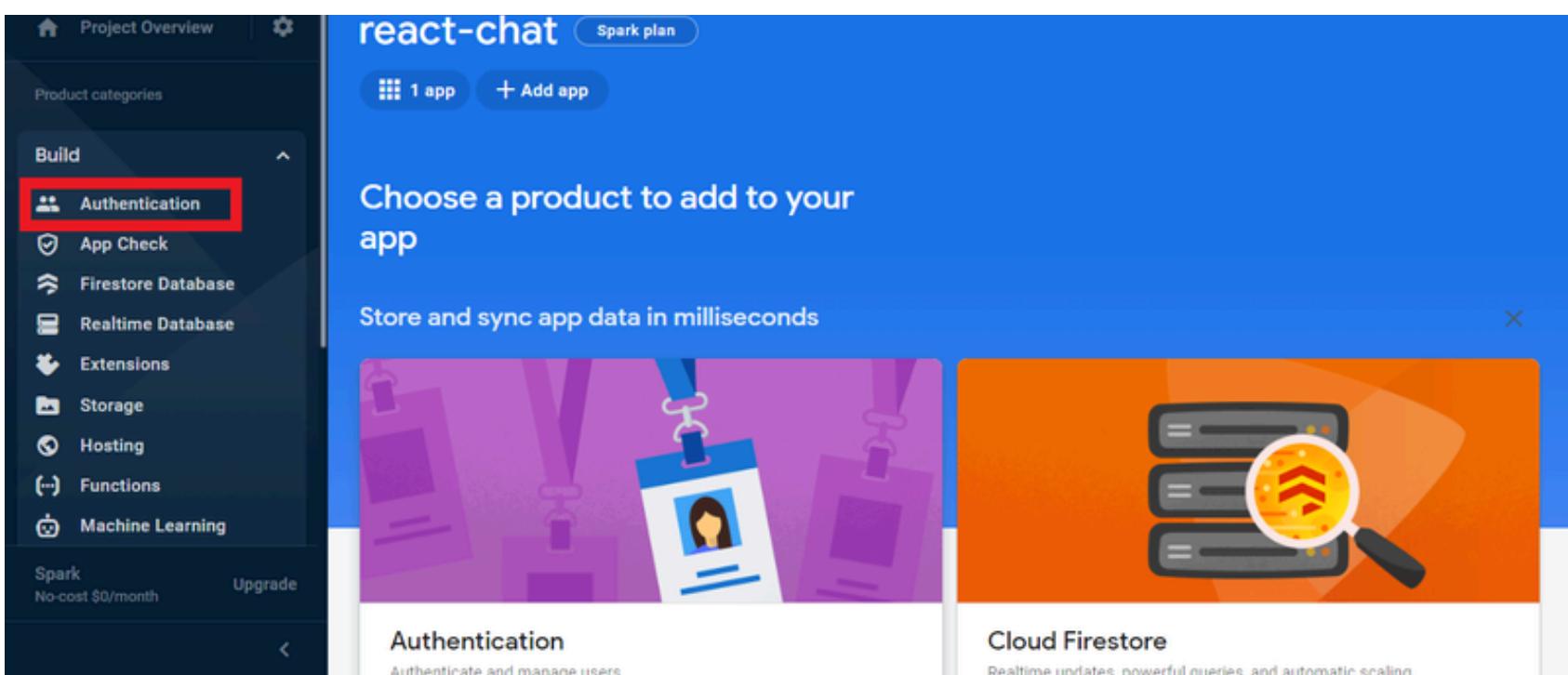
Connect the app to Firebase services for authentication and data storage.

Sets up **getAuth** for user authentication.

Uses **getFirestore** for real-time database.



FIREBASE SETUP



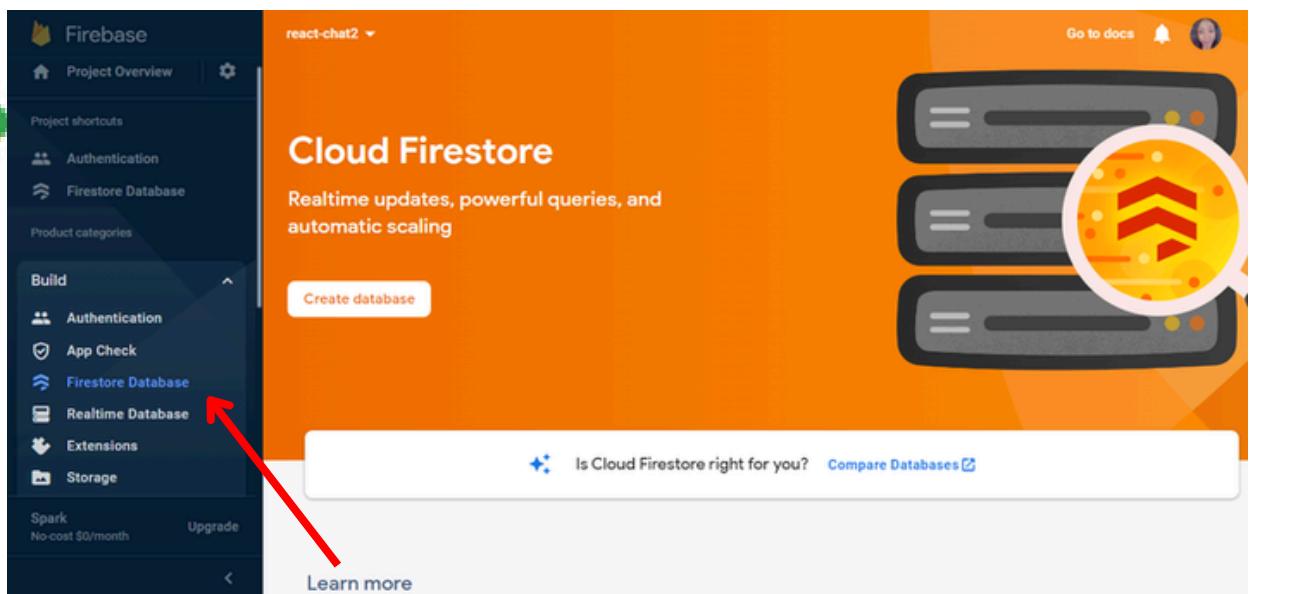
FIREBASE SETUP



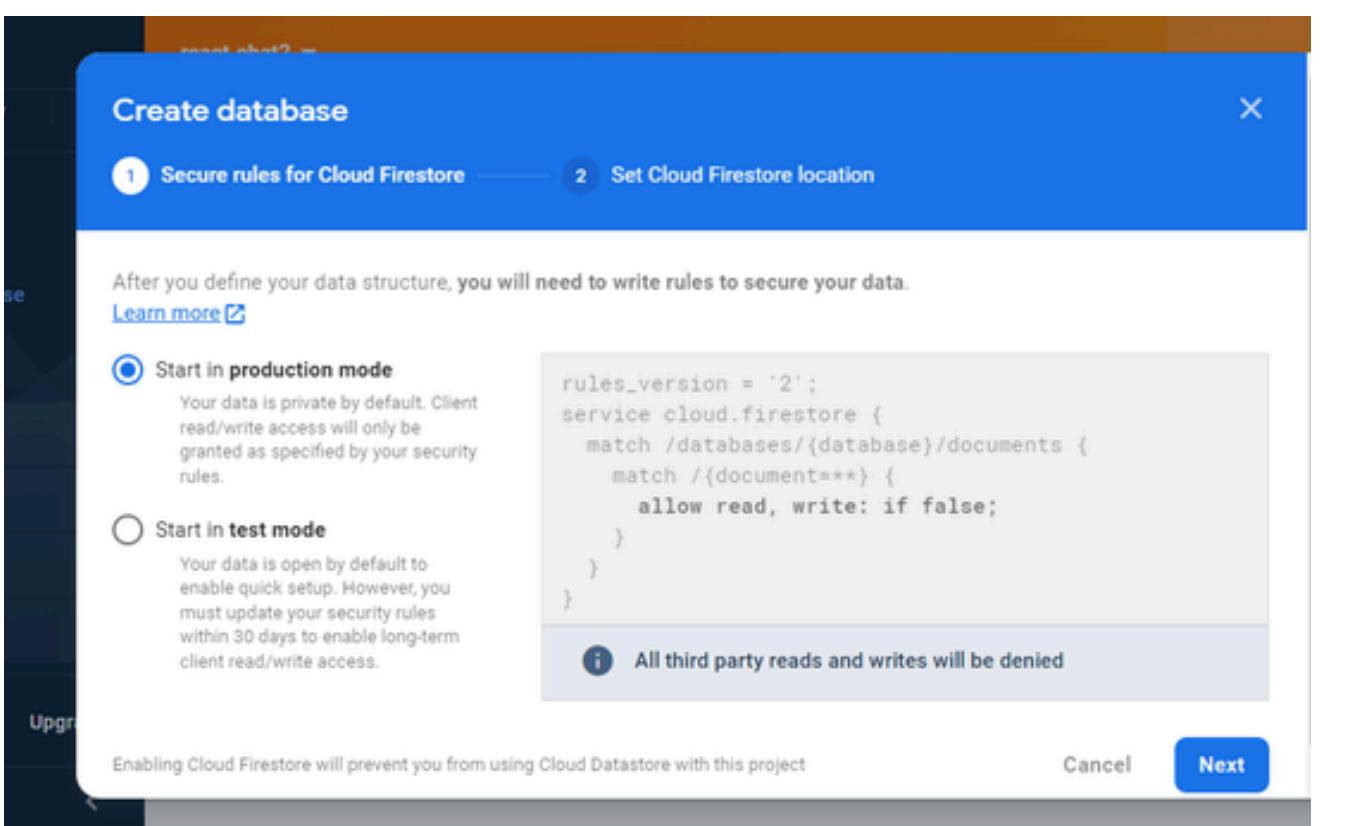
The screenshot shows the 'Authentication' section of a Firebase project. On the left, a sidebar lists 'Project Overview', 'Project shortcuts', 'Authentication' (which is selected), 'Product categories' (Build, Release & Monitor, Analytics, Engage), 'All products', and 'Spark' (No-cost \$0/month). The main area is titled 'Authentication' and has tabs for 'Users', 'Sign-in method' (which is selected), 'Templates', 'Usage', and 'Settings'. Under 'Sign-in providers', there's a message: 'Get started with Firebase Auth by adding your first sign-in method'. It shows 'Native providers' (Email/Password, Phone, Anonymous) and 'Additional providers' (Google, Facebook, Play Games, Game Center, Apple, GitHub, Microsoft, Twitter, Yahoo). A 'Custom providers' section includes OpenID Connect and SAML. A modal window is open on the right, showing 'Product categories' (Build, Release & Monitor, Analytics, Engage) and a 'Sign-in providers' section where 'Google' is selected and enabled.

Choose your authentication sign in method
Enable **Google**, choose your Project support email, and click Save

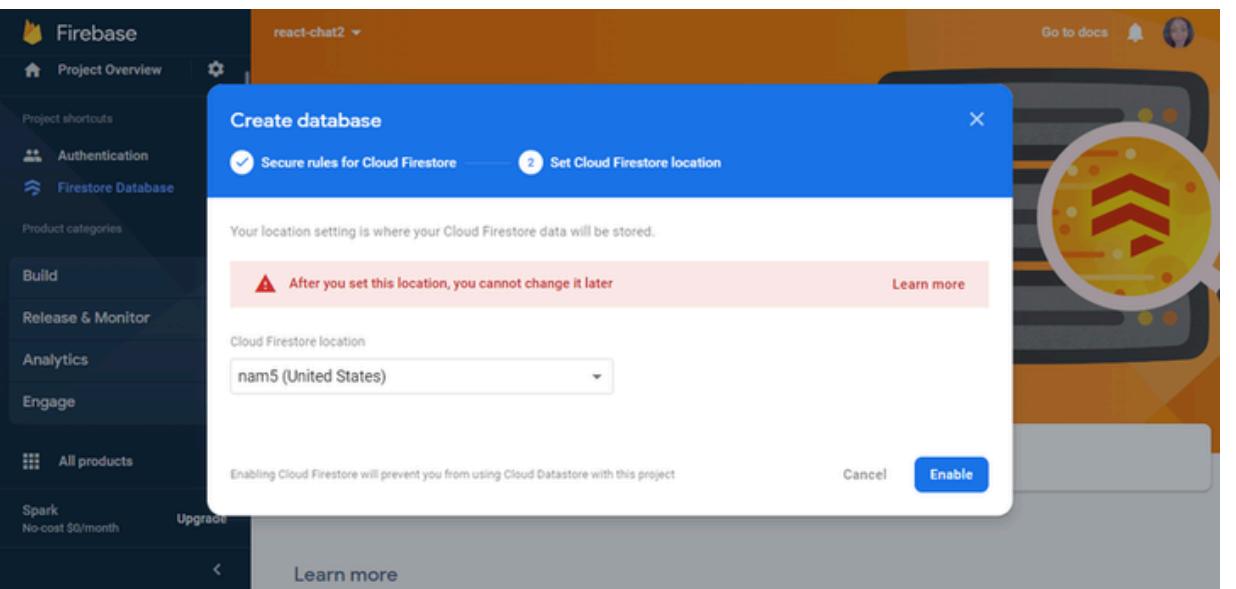
The modal window is titled 'Sign-in providers'. It shows 'Google' is selected and enabled. A note says: 'Google sign-in is automatically configured on your connected Apple and web apps. To set up Google sign-in for your Android apps, you need to add the SHA1 fingerprint for each app on your Project Settings.' Below this, there are fields for 'Project public-facing name' (set to 'Your Project Public Name') and 'Project support email' (set to 'Your Project Email Name'). There are also sections for 'Safelist client IDs from external projects (optional)' and 'Web SDK configuration'. At the bottom are 'Cancel' and 'Save' buttons.



Select **Firestore** in Build Menu



Select **Production Mode** and Next.



Choose for the Region where you use this

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read: if true;
      allow create, update, delete, write: if request.auth != null;
    }
  }
}
```

Before next step, we must Edit our **Firestore rules**



react-chat2

Cloud Firestore

Start a collection

1 Give the collection an ID 2 Add its first document

Parent path /

Collection ID messages

Start a collection

1 Give the collection an ID 2 Add its first document

Document parent path /messages

Document ID 9nC5fXJCZvlUaiu09DP

Field	Type	Value
text	string	hi!

Add field

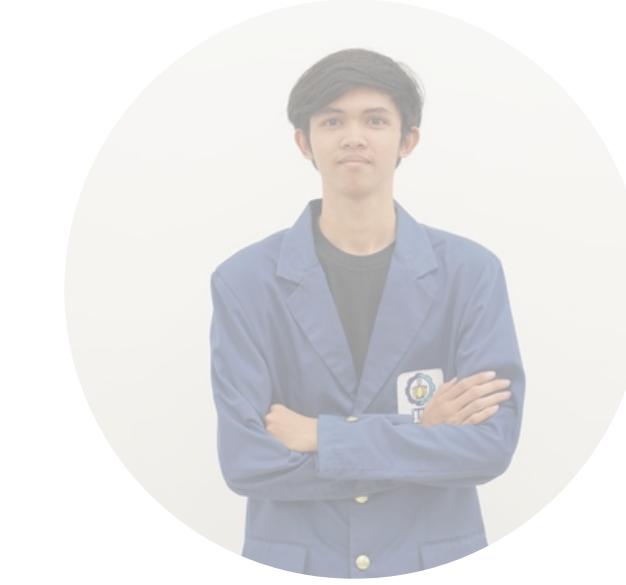
Cancel Save

The screenshot shows the Cloud Firestore 'Start a collection' interface. It displays two steps: 'Give the collection an ID' (step 1) and 'Add its first document' (step 2). Step 1 is active, showing a 'Parent path' of '/' and a 'Collection ID' of 'messages'. Step 2 is shown below it. A modal window is open over the first step, prompting for a 'Document parent path' of '/messages' and a 'Document ID' of '9nC5fXJCZvlUaiu09DP'. Inside the modal, there is a table with a single row showing a field named 'text' of type 'string' with the value 'hi!'. At the bottom of the modal are 'Cancel' and 'Save' buttons.

we can **Start a collection** in Firestore

Last we can create **Collection** in our database
to save our messages request

```
/*1*/  
child: Column(  
crossAxisAlignment: CrossAxisAlignment.  
children: [  
  
/*2*/  
Conta  
pad  
chi  
'  
s  
)  
,  
Text(  
'Ka  
sty  
(  
,
```



THANK YOU

GET IN TOUCH WITH :



AUGISTA_EKSOFON



AUGISTA EKSOFON PRADANA



Google Developer Groups

