Who am I?

Humera Tariq

PhD, MS, MCS (Computer Science), B.E (Electrical)

Postdoc (Medical Image Processing, Deep Neural Networks)

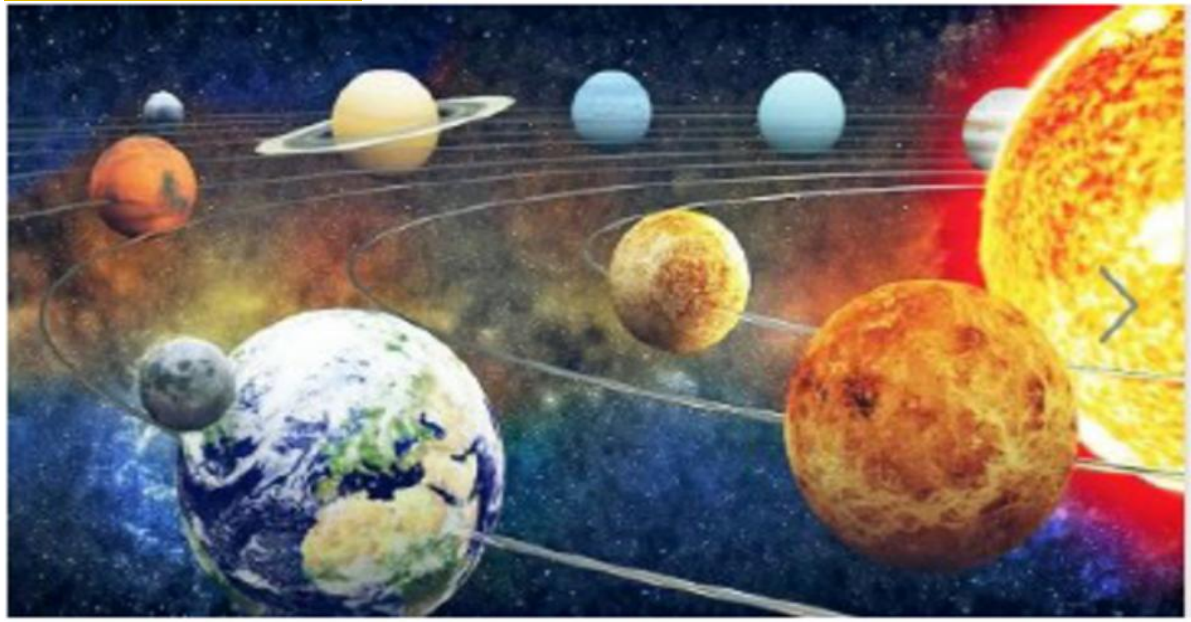
Email: humera@uok.edu.pk

Web: https://humera.pk/

Discord: https://discord.gg/xeJ68vh9

Starting in the name of Allah,

the most beneficial,
the most merciful.



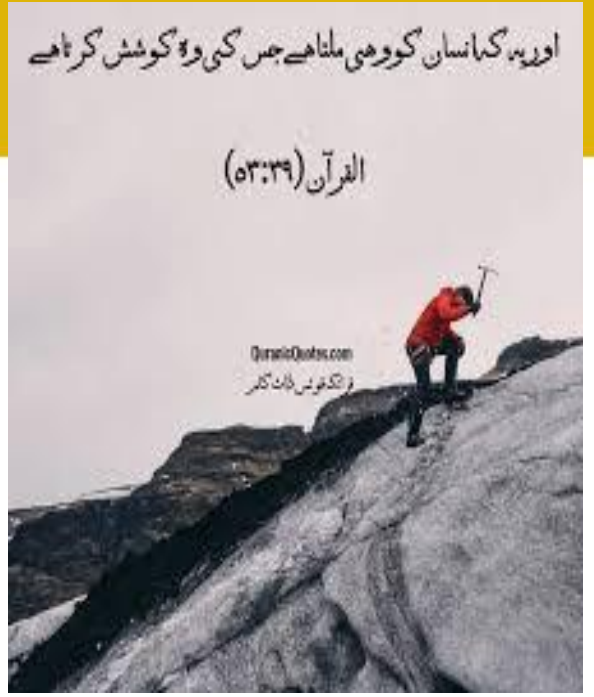
أَمْ لِلْإِنسَانِ مَا تَمَنَّىٰ ﴿٢٤﴾

کیا انسان کو ہر وہ چیز حاصل ہے جس کی اس نے تمنا کی؟

UNIVERSITY OF KARACHI

Surah An-Najm Chapter 53 Verse 39

And there is not for man except that [good] for which he strives.



UNIVERSITY OF KARACHI

Week 10

Internet Application Development

Department of Computer Science (DCS/UBIT)
University of Karachi
January 2025

Week 10 Agenda/ Flow

Firebase

Top React Design Patterns

- ✓ **Example 1 :** Review Mini List Project (addProject, deleteProject, Array of Objects, importance of Backticks)
- ✓ **Example 2 :** Building and hosting a small React app with Fire base authentication. Fire store portion continued.
- ✓ **Example 3 :** Container-Presenter Pattern with clean and optimized fetch logic discussion. (fetch .then vs async await fetch debate).

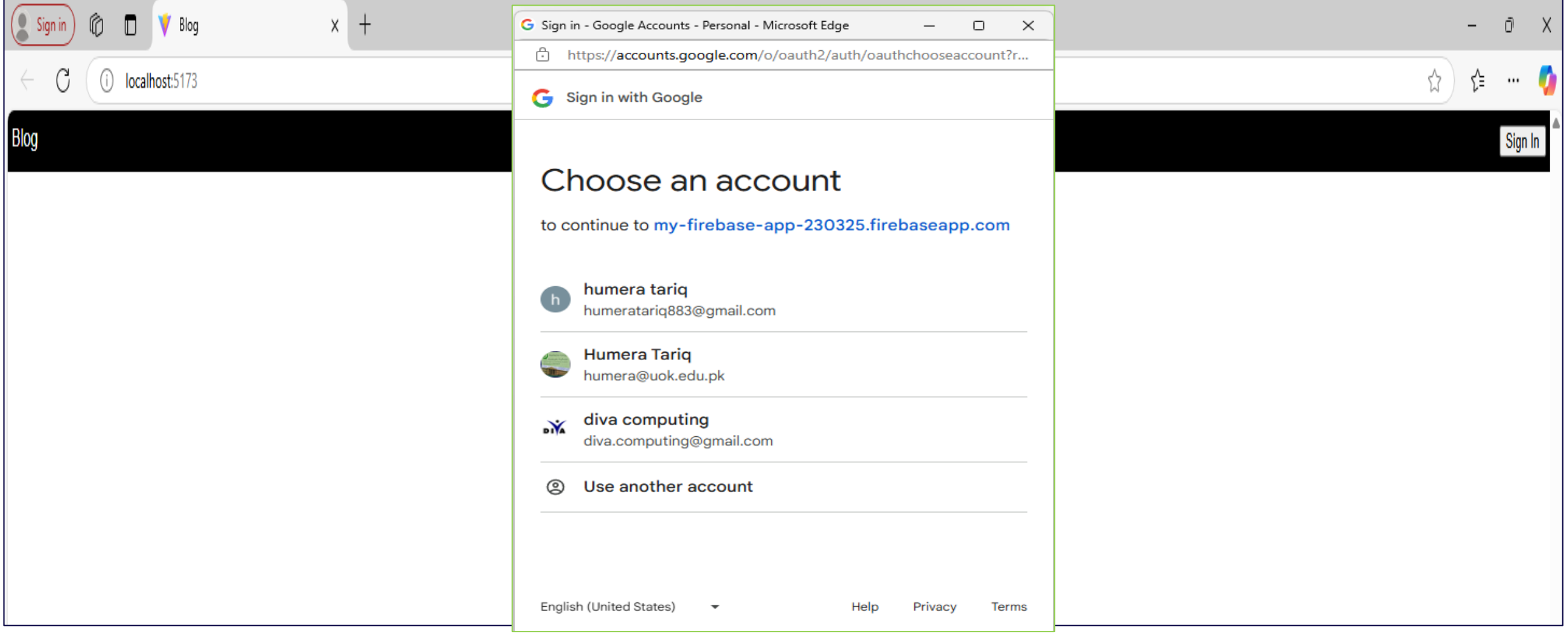
UNIVERSITY OF KARACHI

App.jsx

Step 6: Firestore

Let's talk about some rendering stuff first

Slides 1-30: Flash back week 8-9 +
Form design Assignment


Sign in
Blog
localhost:5173
Blog
Sign In
Sign in - Google Accounts - Personal - Microsoft Edge
https://accounts.google.com/o/oauth2/auth/oauthchooseaccount?r...
Sign in with Google
Choose an account
to continue to my-firebase-app-230325.firebaseapp.com
humera tariq
humeratariq883@gmail.com
Humera Tariq
humera@uok.edu.pk
diva computing
diva.computing@gmail.com
Use another account
English (United States)
Help
Privacy
Terms


localhost:5173
Blog
New Article
Hello, humera tariq
Sign Out
There's a fair tomorrow
Hello Everyone
No article selected

App.jsx


```jsx
return (
  <div className="App">
```

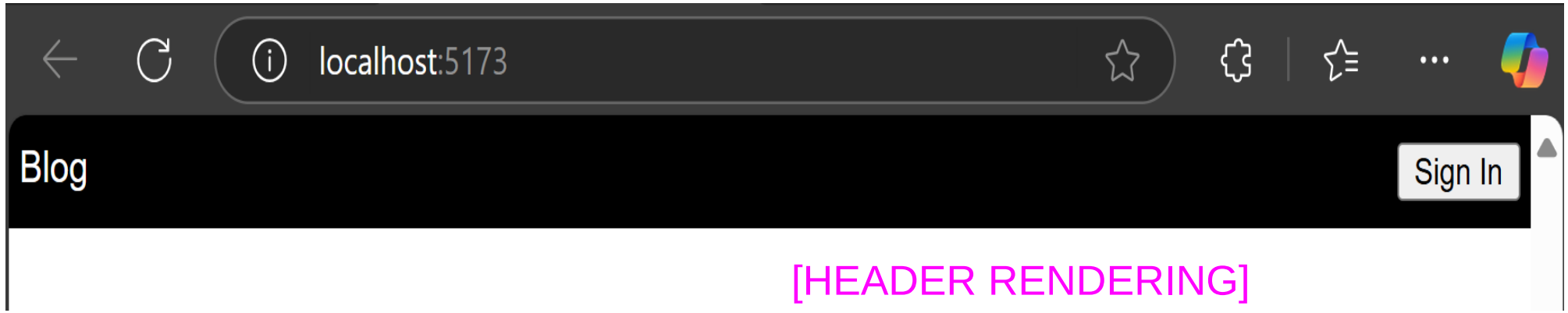
src > components > # App.css > header

```css
body {
  margin: 0;
  font-family: Arial, sans-serif;
}

.App {
  display: grid;
  grid-template-rows: min-content auto;
  grid-template-columns: 250px auto;
  height: 100vh;
  overflow: scroll;
}
```

```css
header {
  display: flex;
  justify-content: space-between;
  background-color: black;
  color: white;
  grid-column: 1 / span 2;
  padding: 10px 5px;
}
```

Can you write JSX / html/Template Literals (InnerHTML) using given condition ?

App.jsx



Describing the UI

[HEADER RENDERING]

Show: "Blog"

Is user?

Show <button> ("New Article")

Show <SignOut />

No user?

Show <SignIn />

localhost:5173

Blog

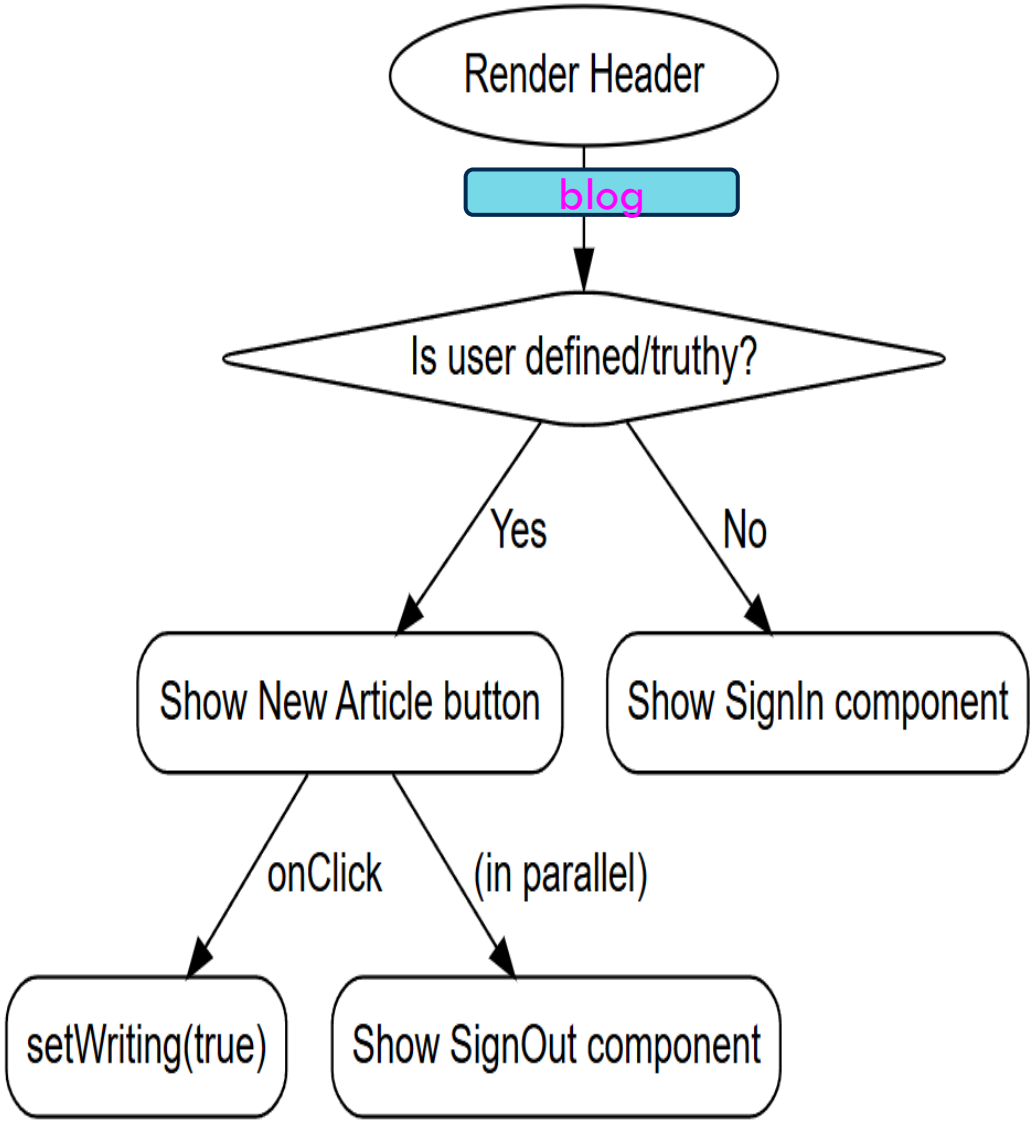
Sign In

Conditional Rendering

Your components will often need to display different things depending on dif

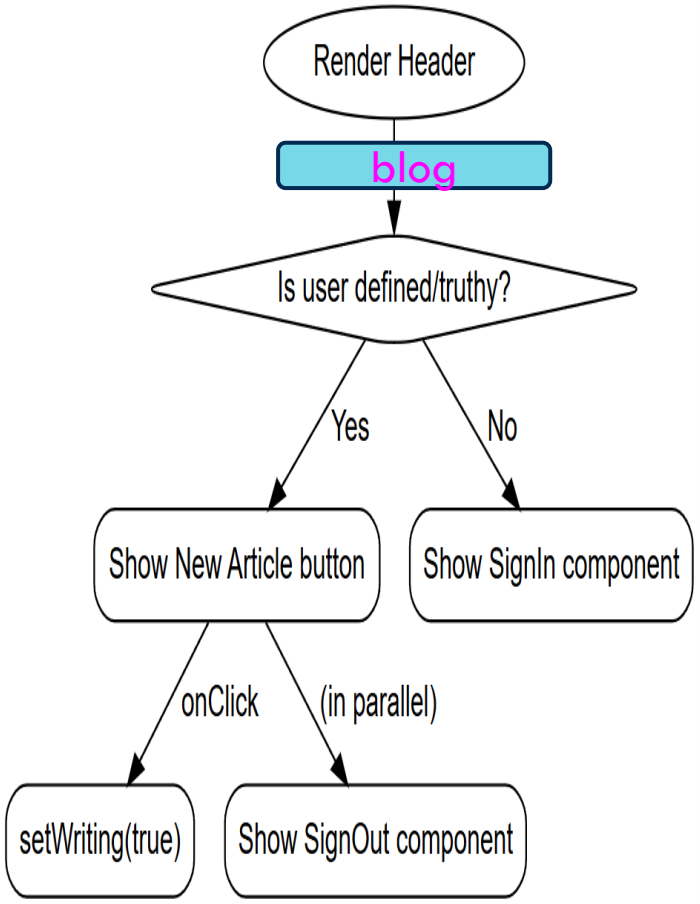
Can you write JSX / html/Template Liter
(InnerHTML) using given condition ?

```
// Writing mode flag state
// Initialized as false, toggles between viewing and writing modes
// When true, UI should show article composition interface
const [writing, setWriting] = useState(false)


const user = useAuthentication()
```



Describing the UI

<header>{….}</header>




```
const [writing, setWriting] = useState(false)
const user = useAuthentication()
```

Conditional rendering

```
<header>
  Blog
  {user && <button onClick={() => setWriting(true)}>New Article</button>}
  {!user ? <SignIn /> : <SignOut />}
</header>
```

<header>{….}</header> //done

How data flows ?

App.jsx -> Nav.jsx

How Data Flows?

App.jsx

```
const [articles, setArticles] = useState([])
const [article, setArticle] = useState(null)
```

|

|--- Manages State

|

|--- JSX Return:

```
<div>
  <header>...</header>
  <Nav articles={articles} setArticle={setArticle} />
  ...
</div>
```

|

|--- Nav.jsx

Receives Props --> Generates HTML

```
<nav>
 ....
</nav>
```

Briefly explain in 2-3 lines that what are you passing as prop to **Nav.jsx** ? And why?

```jsx
export default function App() {

  // Array state to store all articles
    const [articles, setArticles] = useState([])

  // Single article state
  const [article, setArticle] = useState(null)

  // Writing or viewing mode state
  const [writing, setWriting] = useState(false)

  const user = useAuthentication()
```

Can you write a single jsx statement for Nav component with the help of given hints i.e., states, UI and {< />} ?

App.jsx

{<Nav ...??.. />}

```jsx
export default function App() {

  // Array state to store all articles
    const [articles, setArticles] = useState([])

  // Single article state
  const [article, setArticle] = useState(null)

  // Writing or viewing mode state
  const [writing, setWriting] = useState(false)

  const user = useAuthentication()
```

Nav.jsx

Now the question is : What will be returned by **Nav.jsx**

```jsx
export default function Nav({ articles, setArticle })

{

  return (                    ??

  )

}
```

Transform words/flow/UI ->code

```
export default function Nav({
 articles, setArticle })
{
```

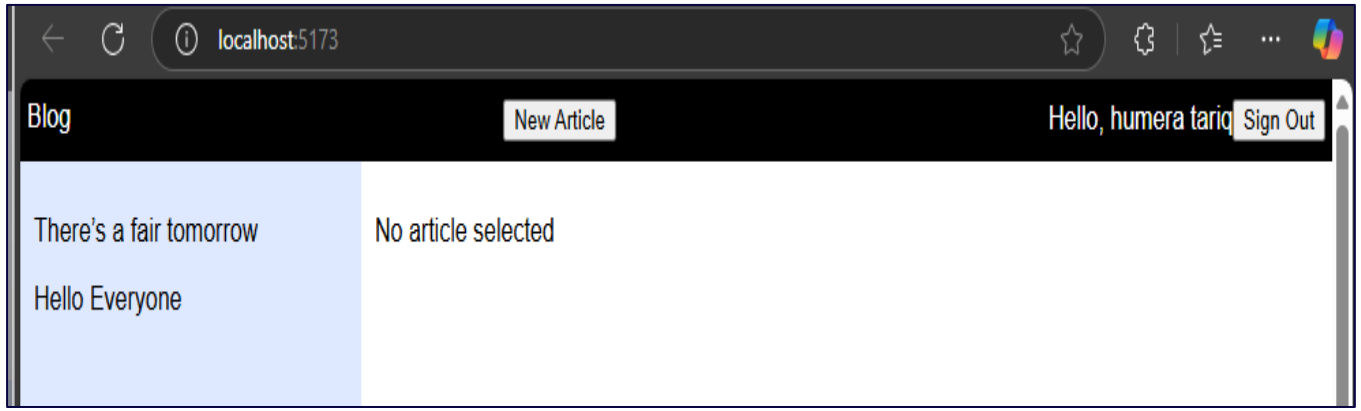
The Nav component conditionally renders either:

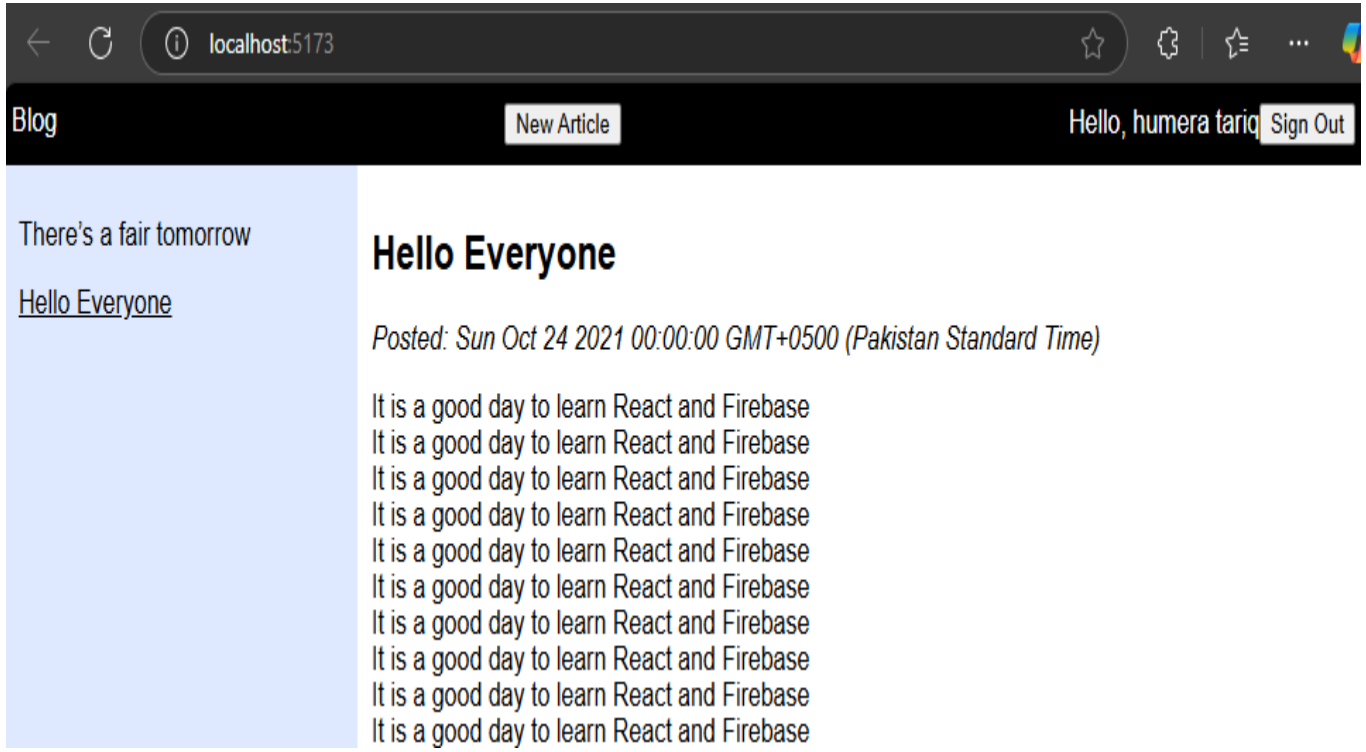
- A message "No articles" if the array is empty,

OR

- A list of clickable <p> elements
  - — each showing an article's title
  - — and on click, the corresponding article is selected via setArticle(a).

```
}
```

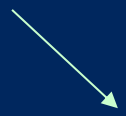




Comparing with project List ?

App.jsx -> Nav.jsx -> ArticleEntry ->Article

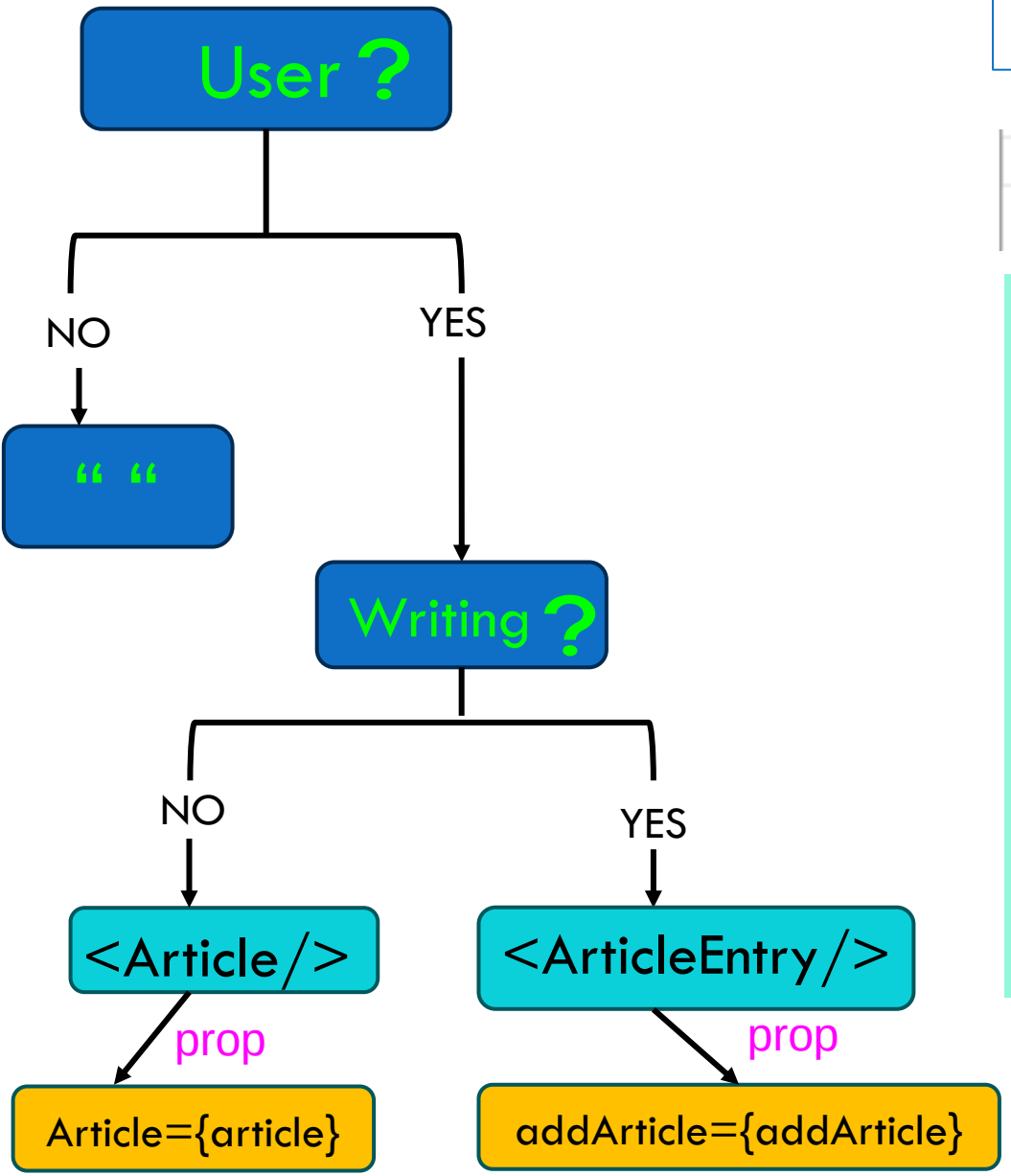
App.jsx -> ProjectList.jsx -> ProjectEntryForm-> projects



DeleteFromList.jsx

Transform the flow to rendering code

App.jsx




```
const [articles, setArticles] = useState([])
const [article, setArticle] = useState(null)
```

```
const [writing, setWriting] = useState(false)
const user = useAuthentication()
```

```
{!user ? (
        ""
    ) : writing ? (
   <ArticleEntry addArticle={addArticle} />
    ) : (
      <Article article={article} />
    )}
```

Rendering is still on?

Dataflow is still on?

App.jsx -> Nav.jsx

ArticleEntry.jsx -> Article.jsx

“We realized that ArticleEntry is a form.

ArticleEntry.jsx

Form Component in React is any component that:

-Collects user input - Handles input events (onChange) -validates input -Submits data (onSubmit handler)

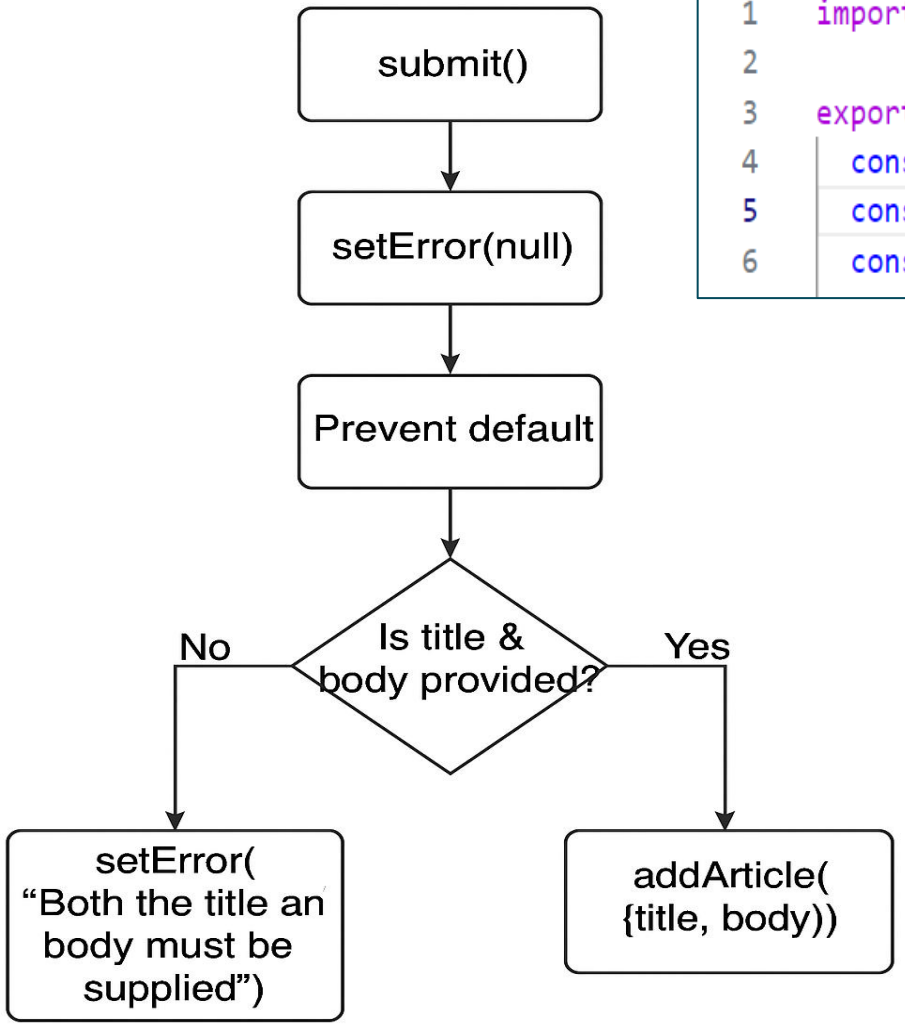
```
<div>
└── <form>
    ├── {error && <p>} (conditional)
    ├── Title Section
    │   ├── "Title" (text node)
    │   └── <input>
    ├── Body Section
    │   ├── "Body" (text node)
    │   └── <textarea>
    └── <button>
```

```
// Controlled input: value is set by state
                     e.g. input value={title}
```

src > components > ArticleEntry.jsx > ArticleEntry

```jsx
import { useState } from "react"

export default function ArticleEntry({ addArticle }) {
  const [title, setTitle] = useState("")
  const [body, setBody] = useState("")
  const [error, setError] = useState(null)
```

ArticleEntry.jsx




```
src > components > ArticleEntry.jsx > ArticleEntry
1   import { useState } from "react"
2
3   export default function ArticleEntry({ addArticle }) {
4     const [title, setTitle] = useState("")
5     const [body, setBody] = useState("")
6     const [error, setError] = useState(null)
```

```
function submit(e) {
    setError(null)
    e.preventDefault()
    if ( ____________________ ) {
      setError("Both the title and body must be supplied")
    } else {

    }
  }
```

"Lifting State Up" or *"Callback Props Pattern"*

ArticleEntry.jsx

Pass *function* down → Child *calls it* → Parent is *notified*.

[Parent Component]
|
|--- pass down addArticle() in props
|
[Child Component]
|
|--- Calls addArticle() on submit
|
[Parent Handles State Change]

```jsx
// Update the "database" *then* update the internal React state. These
// two steps are definitely necessary.
function addArticle({ title, body }) {
  createArticle({ title, body }).then((article) => {
    setArticle(article)
    setArticles([article, ...articles])
    setWriting(false)
  })
}
```

App.jsx

```jsx
{!user ? ("") : writing ? (
        <ArticleEntry addArticle={addArticle} />
      ) : (
        <Article article={article} />
      )}
```

ArticleEntry.jsx

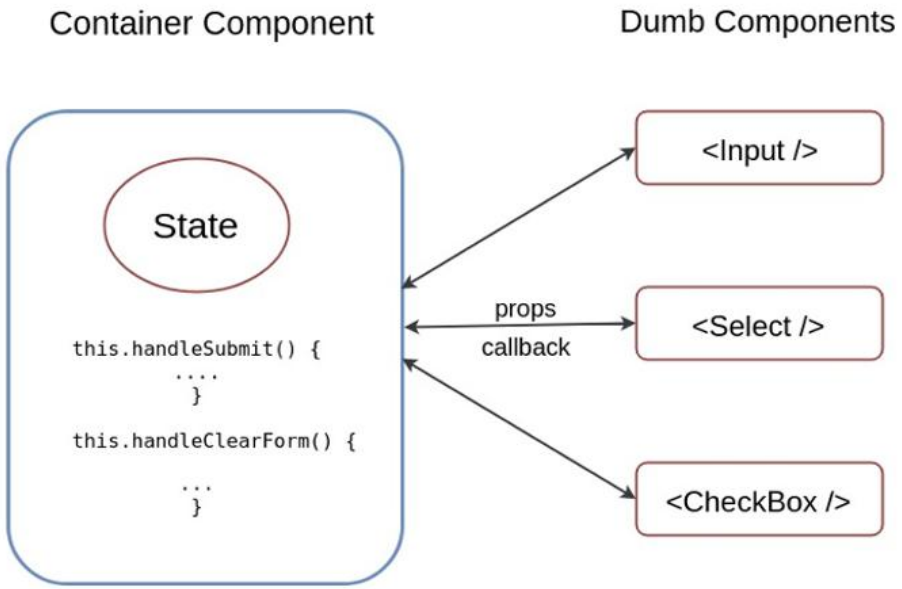
The form data, for instance, is usually handled by the component rather than the DOM and is usually implemented using controlled components.

Details of Element	Purpose	React Concept
<form onSubmit={submit}>	HTML Form Element	Form Submission Handling
<input value={title} onChange={...} />	Controlled Input	Controlled Component
<textarea value={body} onChange={...} />	Controlled Input	Controlled Component
submit() function	Form Validation + Submit Logic	Event Handler
addArticle({ title, body })	Pass Data to Parent	Lifting State up
setError()	Show Error to User	Conditional Rendering

Controlled components working in React

ArticleEntry.jsx

The form's structure is similar to those of the usual HTML forms. However, *each input element gets a component of its own*, which we call dumb components.

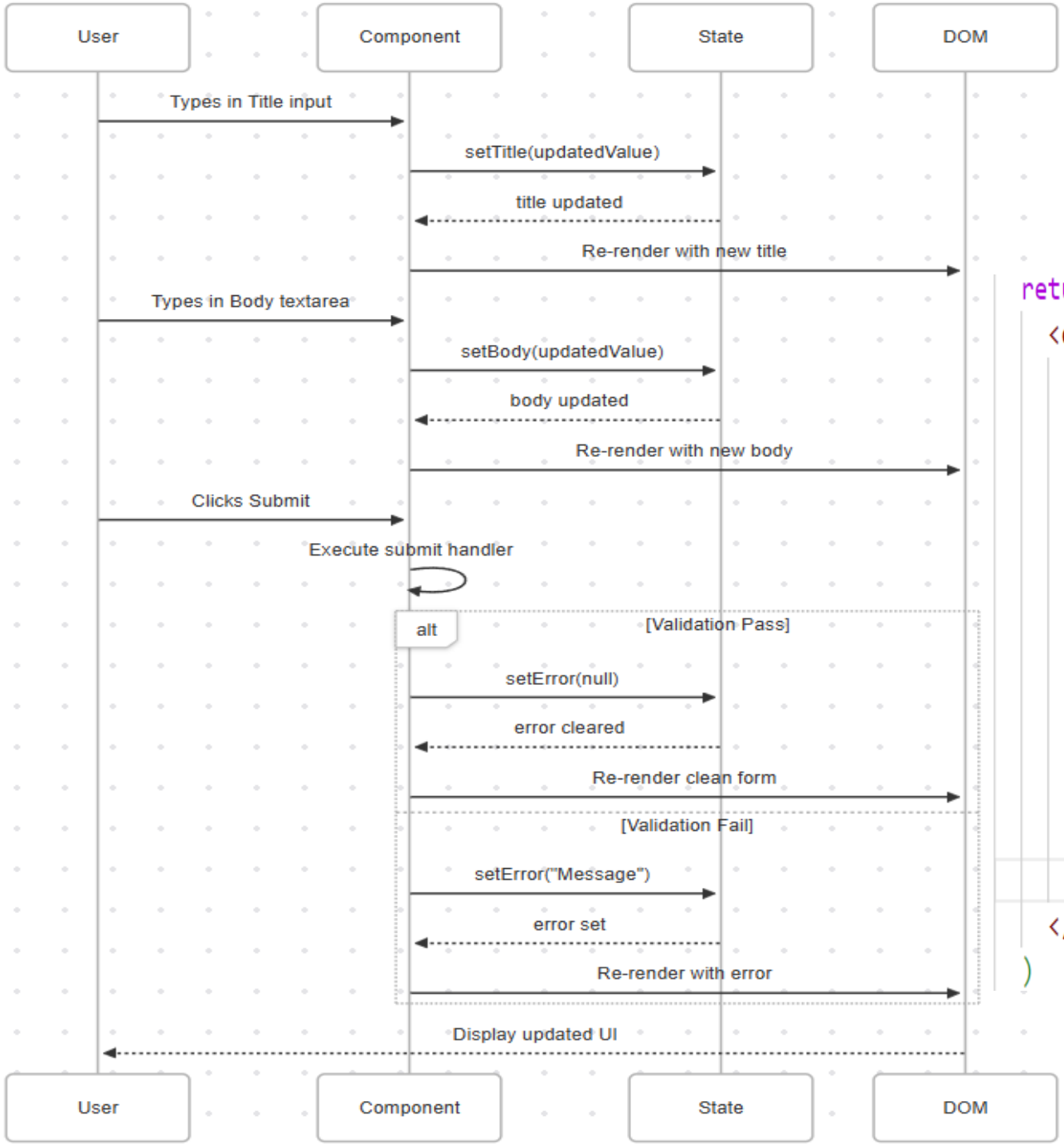



```
// onChange updates the state
```

src > components > ArticleEntry.jsx > ArticleEntry

```
1   import { useState } from "react"
2
3   export default function ArticleEntry({ addArticle }) {
4     const [title, setTitle] = useState("")
5     const [body, setBody] = useState("")
6     const [error, setError] = useState(null)
```

Briefly explain the rendering sequence of ArticleEntry.jsx




```jsx
return (
  <div>
    <form onSubmit={submit}>
      {error && <p className="error">{error}</p>}
      Title
      <input value={title} onChange={(e) => setTitle(e.target.value)} />
      Body
      <textarea
        rows="8"
        value={body}
        onChange={(e) => setBody(e.target.value)}
      ></textarea>
      <button type="submit">Create</button>
    </form>
  </div>
)
```

inputType={"text"}

inputType={"number"}

<Select />

<CheckBox />

<TextArea />

<Button/>

Sample Form Container

Full Name

Enter your name

Age

Enter your age

Gender

Select Gender

Skills

Programming Development Design Testing

About you.

Describe your past experience and skills

Submit Clear

Component Article

App.jsx -> Nav.jsx

ArticleEntry.jsx -> Article.jsx

Article.jsx

src > components > Article.jsx > Article

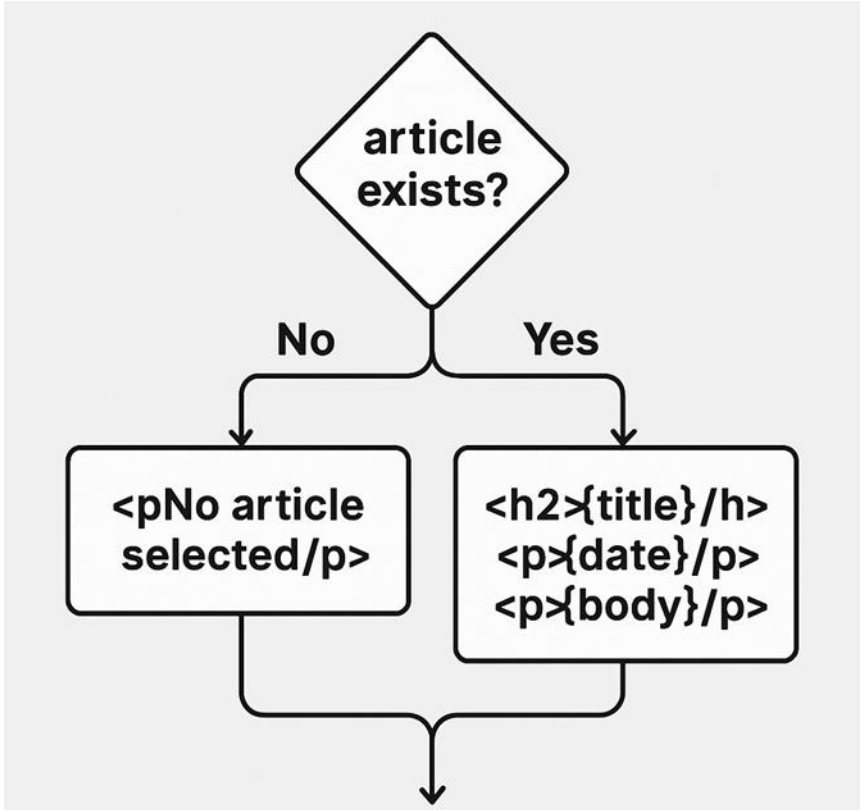
```jsx
export default function Article({ article }) {
  return (
    <article>
      {!article ? (
        <p>No article selected</p>
      ) : (
        <section>
          <h2>{article.title}</h2>
          <p className="date">{`Posted: ${article.date}`}</p>
          <p className="body">{article.body}</p>
        </section>
      )}
    </article>
  )
}
```

Transform .jsx to flow chart

Article.jsx

src > components > Article.jsx > Article

```jsx
export default function Article({ article }) {
  return (
    <article>
      {!article ? (
        <p>No article selected</p>
      ) : (
        <section>
          <h2>{article.title}</h2>
          <p className="date">{`Posted: ${article.date}`}</p>
          <p className="body">{article.body}</p>
        </section>
      )}
    </article>
  )
}
```



App.jsx

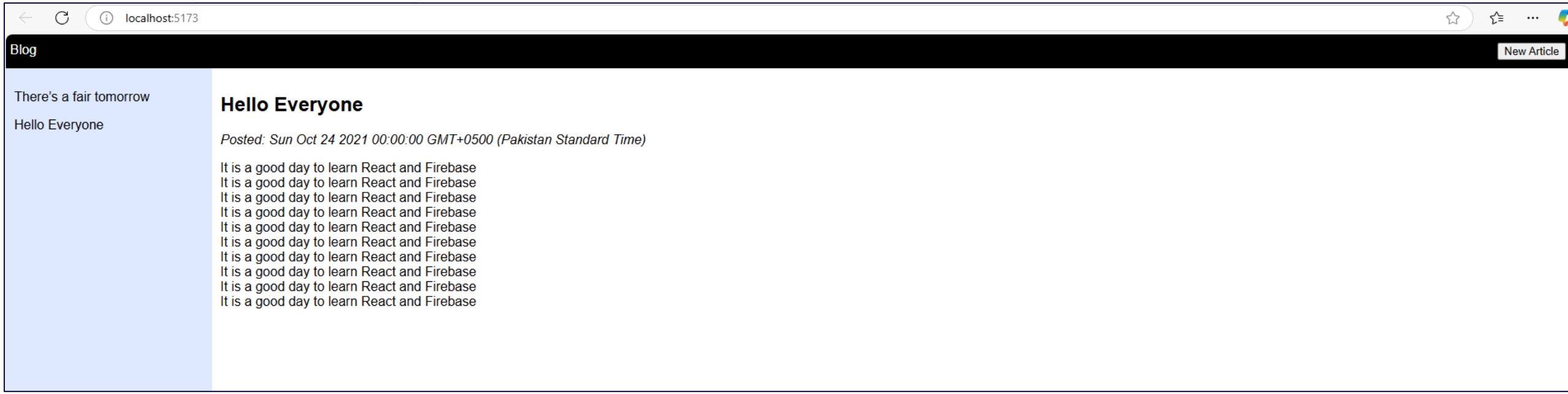
Step 6: Firestore

DB

Get the Starter Code (i.e., without Firebase/Firestore)

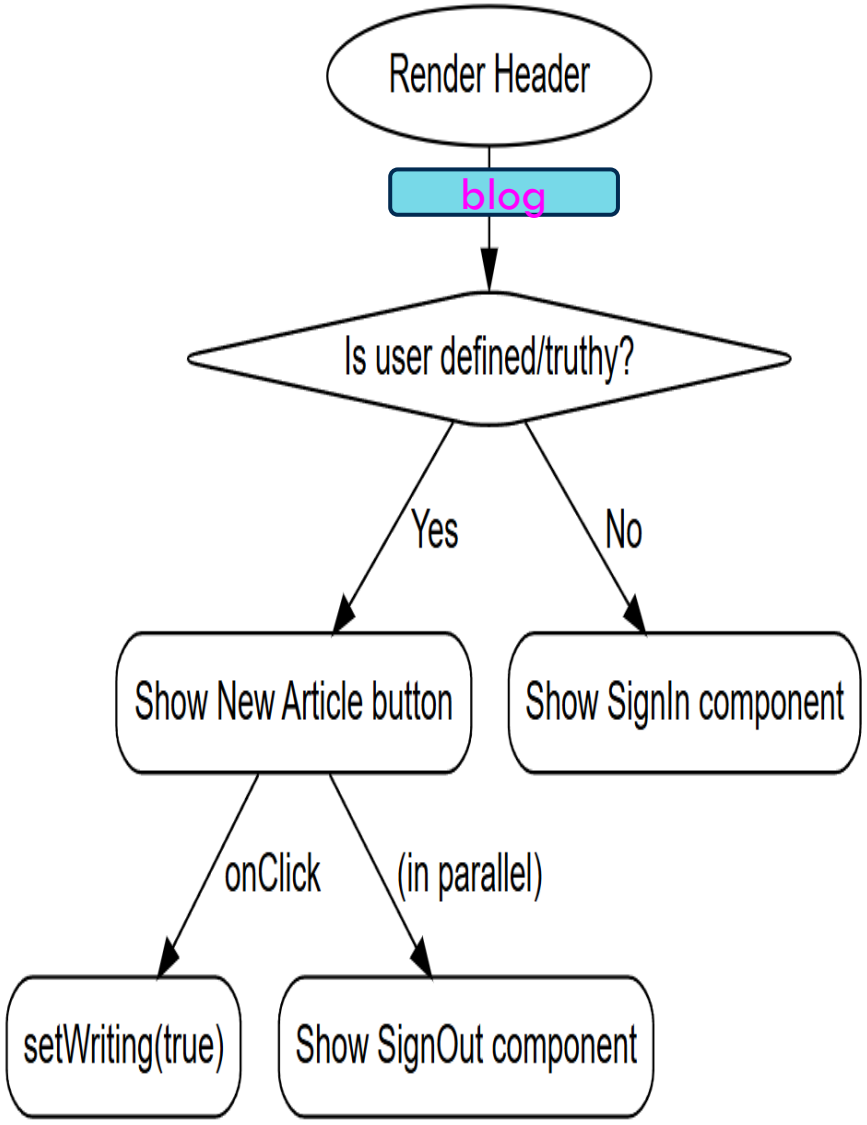
Get the Starter code

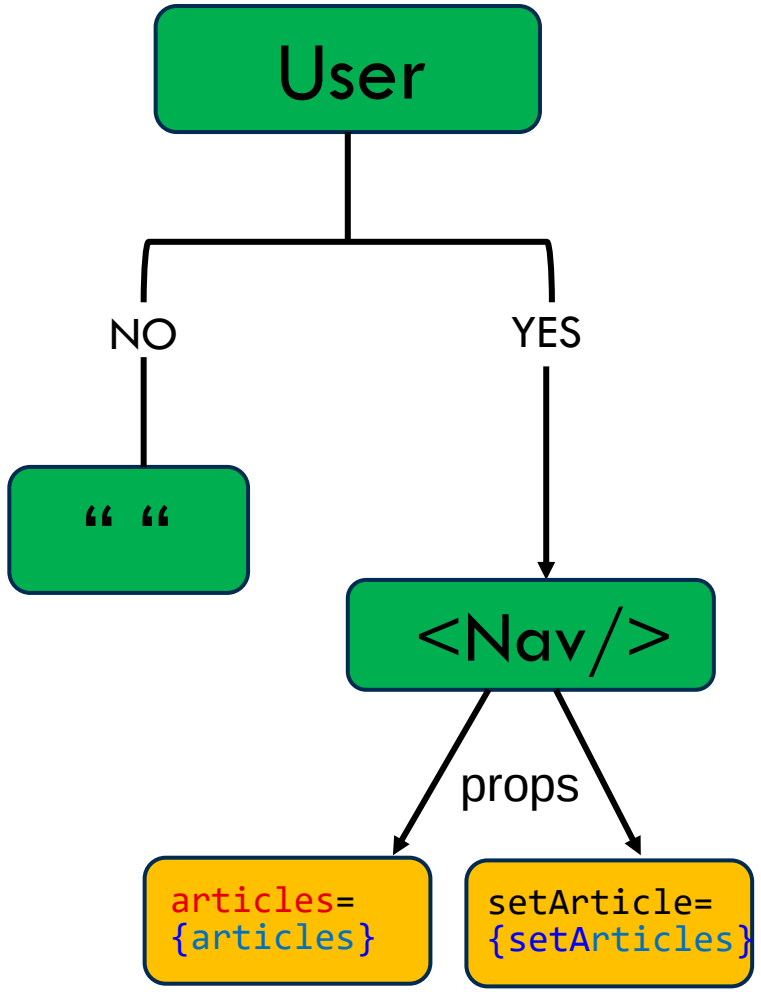
```
├── README.md
├── eslint.config.js
├── vite.config.js
├── index.html
├── package-lock.json
├── package.json
└── src
    ├── main.jsx
    ├── components
    │   ├── App.css
    │   ├── App.jsx
    │   ├── Article.jsx
    │   ├── ArticleEntry.jsx
    │   └── Nav.jsx
    └── services
        └── articleService.js
```

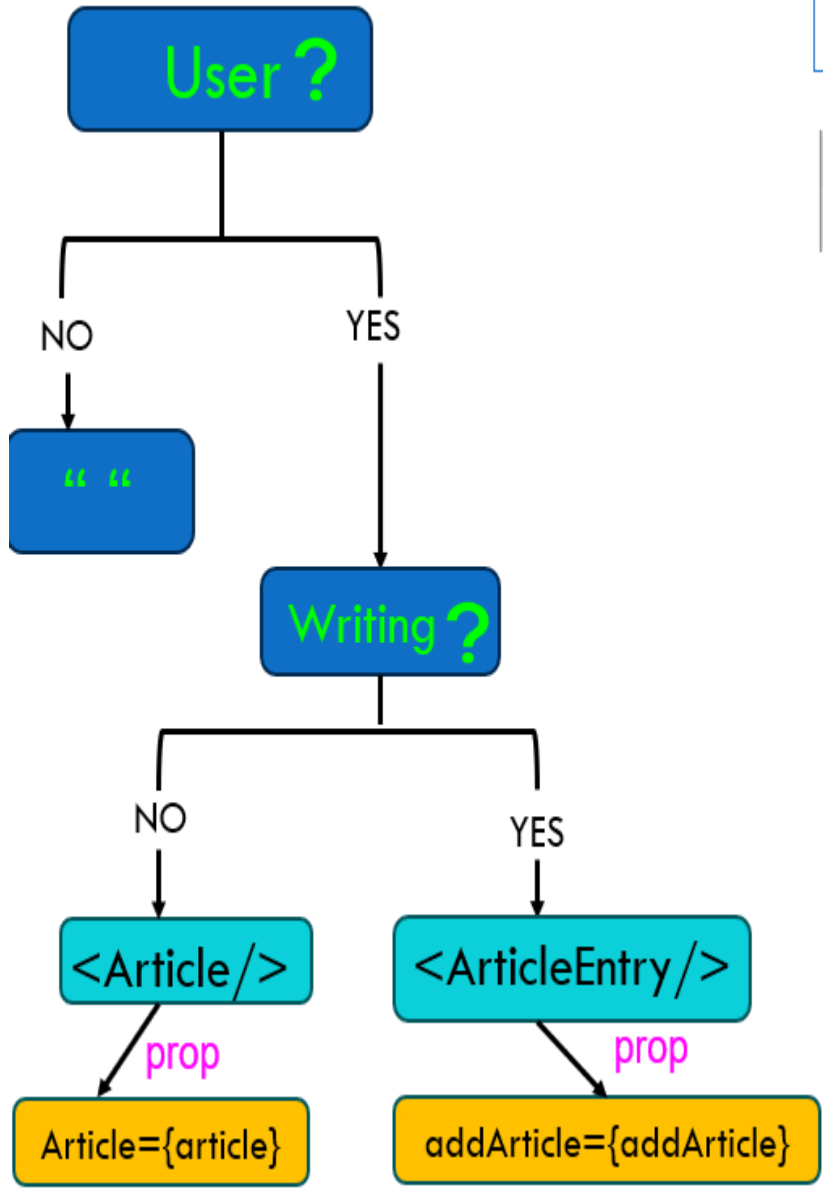


Conditional Rendering

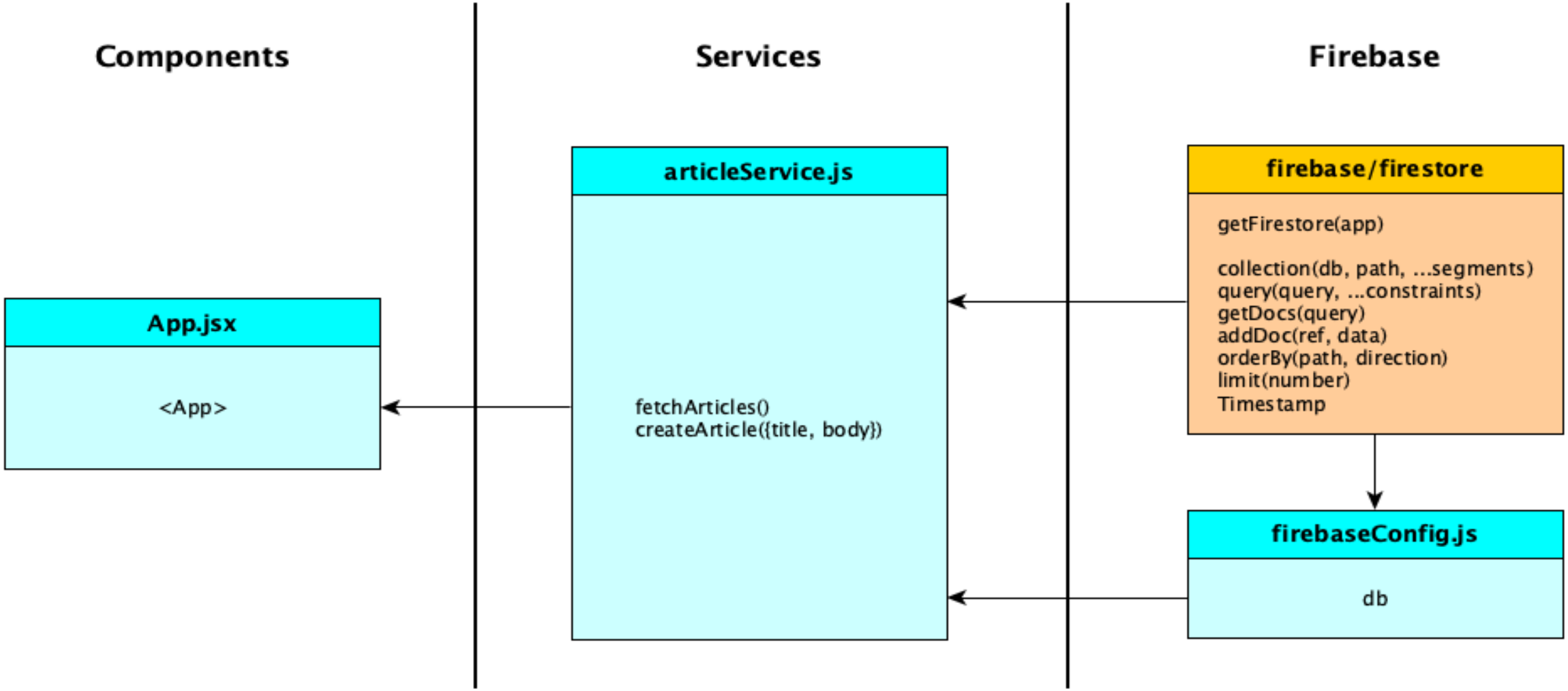
App.jsx





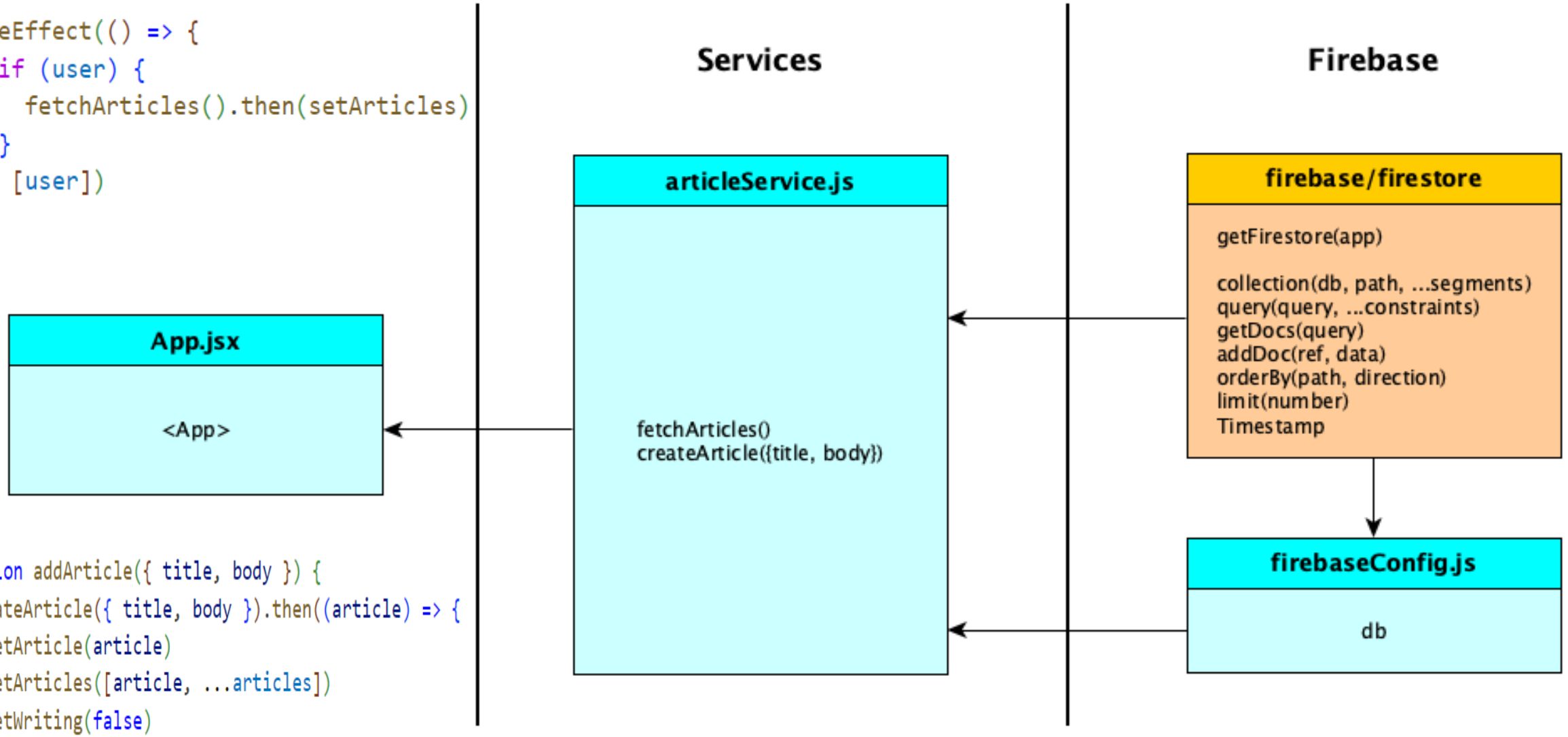


This time we want to integrate Firestore to replace the mock blog data with data from a real database.



```
import { fetchArticles, createArticle } from "../services/articleService"
```

```
useEffect(() => {
  if (user) {
    fetchArticles().then(setArticles)
  }
}, [user])
```

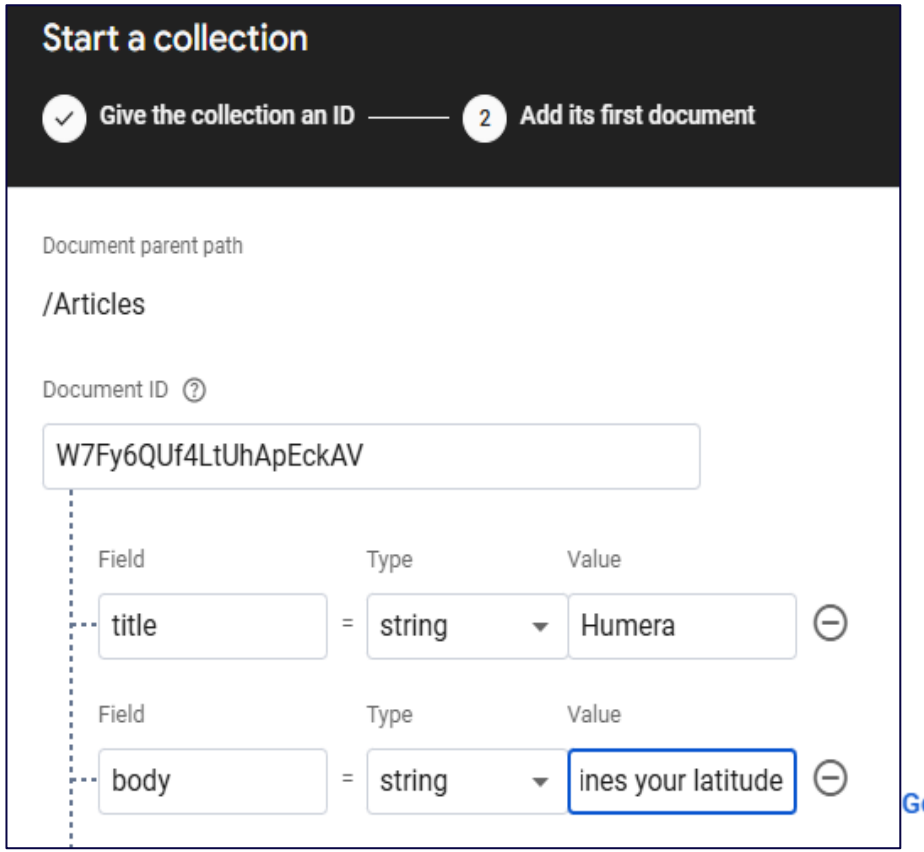


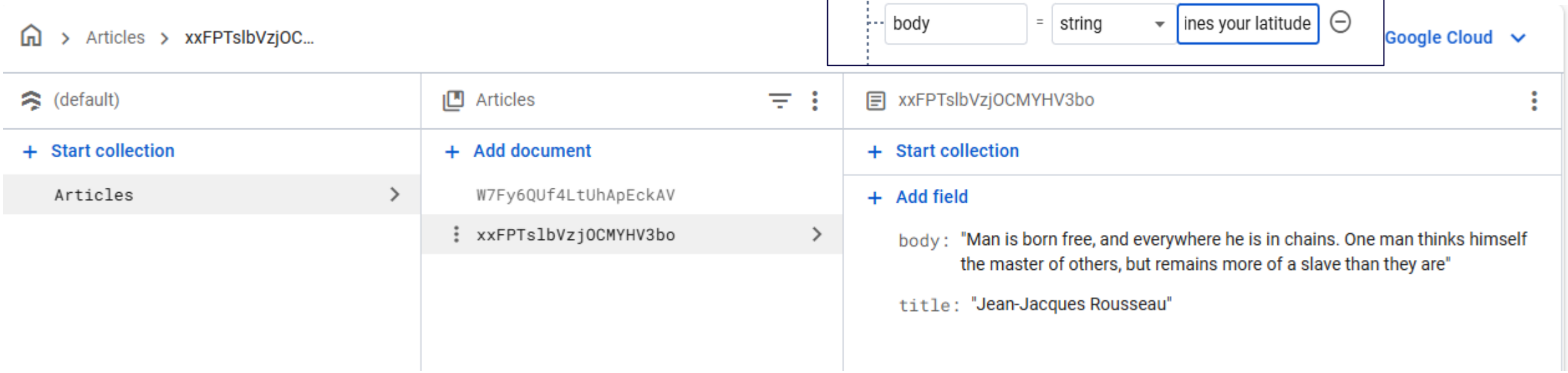

```
function addArticle({ title, body }) {
  createArticle({ title, body }).then((article) => {
    setArticle(article)
    setArticles([article, ...articles])
    setWriting(false)
  })
}
```

Time to replace the mock blog data with data from a real database.

In the Firebase Console, add a few documents into a new collection called articles. Each document should have fields title (of type string), body (of type string), and date (of type timestamp). Let Firebase choose the ids.

It's time to use the database instead of the fake data.

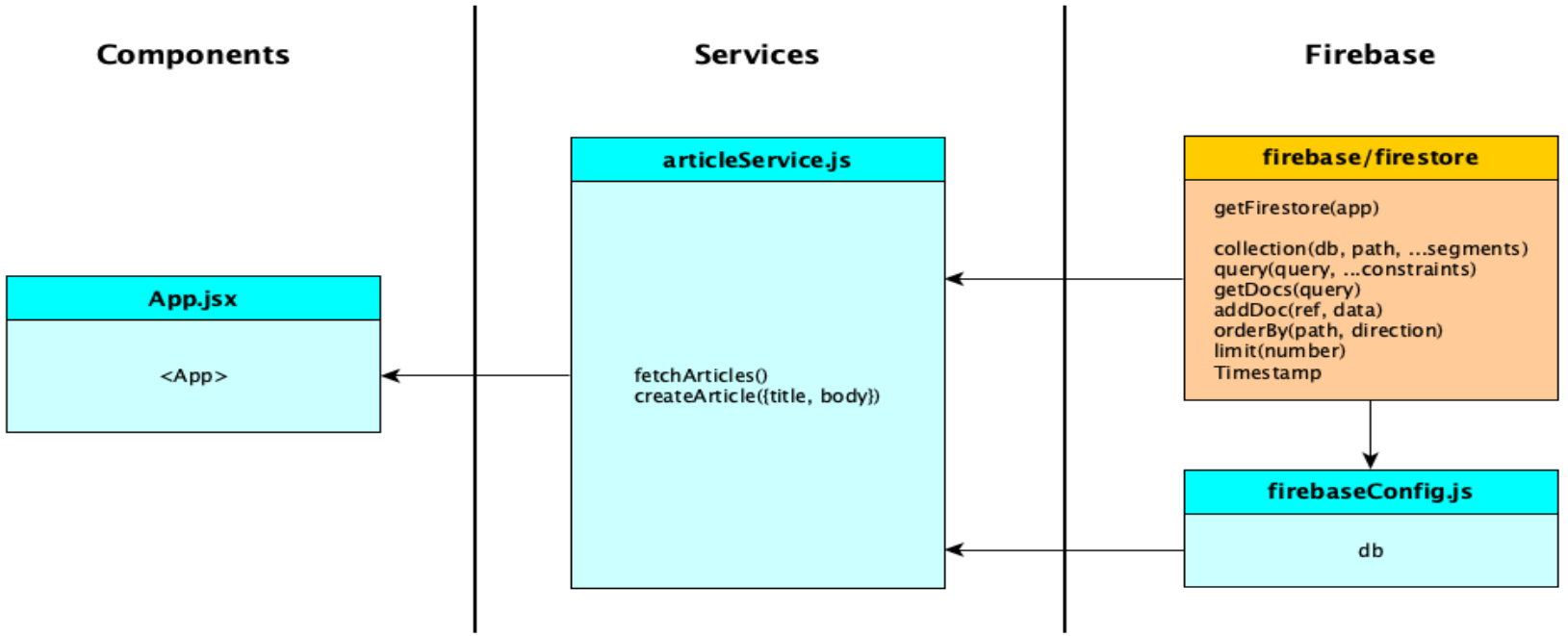




```
import { fetchArticles, createArticle } from "../services/articleService"
```

we have an article service in the starter code, but it used mock articles like in our project web app

```
.
├── README.md
├── eslint.config.js
├── vite.config.js
├── index.html
├── package-lock.json
├── package.json
└── src
    ├── main.jsx
    ├── components
    │   ├── App.css
    │   ├── App.jsx
    │   ├── Article.jsx
    │   ├── ArticleEntry.jsx
    │   └── Nav.jsx
    └── services
        └── articleService.js
```



We will have an article service that hides the complexities of Firestore from the rest of the app, so that the components just make simple calls like fetchArticles and createArticle

articles

- sdkjfhshfskdlshjf
  - date: "2021-03-16"
  - title: "There's a fair tomorrow"
  - body: "Is a mháithrín an ligfidh tú chun aonaigh mé\n..."
- asjkdhalfkjsdjfhsd
  - date: "2021-10-24"
  - title: "Hello Everyone"
  - body: "It is a good day to learn React and Firebase..."

```
export async function fetchArticles() {
  // In storage the ids are separated from the data, but in this function
  // we are going to combine the id and the data together.
  return Object.entries(articles).map(([id, data]) => ({ id, ...data }))
}
```

```
export async function createArticle({ title, body }) {
  // As this is just fake data for messing around, we'll throw in a quick
  // and unreliable database id. In a real app, the id should be generated
  // by the database itself (or you can use UUIDs).
  return { id: Math.random(), title, body, date: new Date() }
}
```

src/services/articleService.js

articles is your fake in-memory storage (key-value store)

```
const articles = {
    asjkdhalfkjsdjfhsd: {
    date: new Date(2021, 9, 24),
    title: "Hello Everyone",
    body: "It is a good day to learn React and Firebase\n".repeat(10),
  },
}
```

```
export async function createArticle({ title, body }) {
  // As this is just fake data for messing around, we'll throw in a quick
  // and unreliable database id. In a real app, the id should be generated
  // by the database itself (or you can use UUIDs).
  return { id: Math.random(), title, body, date: new Date() }
}
```

"Right now: No Network. No API. Just Memory."
"Later: Real Network. Real API. Real Database."

- ✓ No HTTP call
- ✓ No server
- ✓ No database
- ✓ No fetch()
- ✓ No axios()
- ✓ No network delay

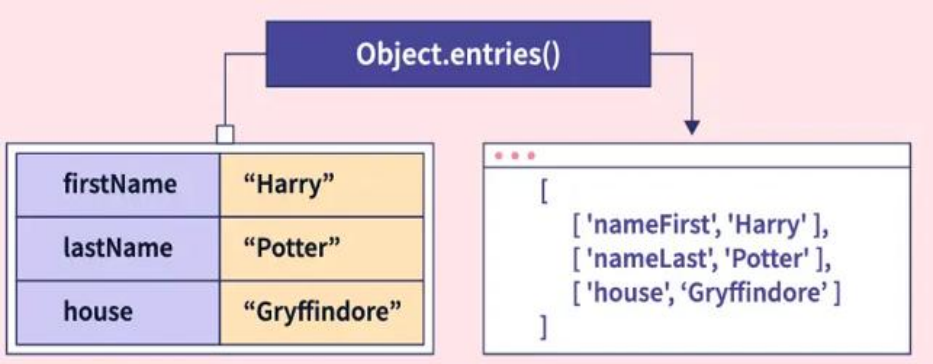
"Local memory is instant.

Network request is slow because it leaves your computer."

Transforming Data Storage for Frontend Use

Local fetch discussion

src/services/articleService.js



Converts an object into an array of [key, value] pairs.

```
articles = {
   id1: { title: ..., body: ..., date: ... },
   id2: { title: ..., body: ..., date: ... },
}
```

```
[
   [id1, { title: ..., body: ..., date: ... }],
   [id2, { title: ..., body: ..., date: ... }]
]
```

```
export async function fetchArticles() {
  // In storage the ids are separated from the data, but in this function
  // we are going to combine the id and the data together.
  return Object.entries(articles).map(([id, data]) => ({ id, ...data }))
}
```

```
export async function fetchArticles() {
  // In storage the ids are separated from the data, but in this function
  // we are going to combine the id and the data together.
  return Object.entries(articles).map(([id, data]) => ({ id, ...data }))
}
```

array of [key, value] pairs

```
[
   [id1, { title: ..., body: ..., date: ... }],
   [id2, { title: ..., body: ..., date: ... }]
]
```

Array.map()

- **Purpose:** Transform each array item. Immutable operation (creates a new array).
- **Destructuring:** ([id, data]) extracts the key (id) and value (data) from each entry.

forEach() vs map()

Understanding the key differences between array iteration methods

.forEach()

Executes a function for each array element. Does not create a new array and returns undefined.

```
const numbers = [1, 2, 3];
const result = numbers.forEach(
num => console.log(num * 2)
);
// Logs: 2, 4, 6
// result = undefined
```

.map()

Creates a new array with the results of calling a function for each array element.

```
const numbers = [1, 2, 3];
const result = numbers.map(
num => num * 2
);
// result = [2, 4, 6]
```

Key Differences

- **Return Value:** forEach returns undefined, map returns a new array
- **Chaining:** map can be chained with other methods, forEach cannot
- **Memory:** map creates a new array, forEach doesn't allocate additional memory
- **Purpose:** forEach for side effects, map for transforming data
- **Performance:** forEach slightly faster when not needing return values

When to Use Each

Use forEach() when

- You're not creating a new array
- Performing operations with side effects
- Just need to iterate over elements

Use map() when

- You need a new array based on another array
- Transforming data
- Need to chain operations

```
export async function fetchArticles() {
  // In storage the ids are separated from the data, but in this function
  // we are going to combine the id and the data together.
  return Object.entries(articles).map(([id, data]) => ({ id, ...data }))
}
```

Final output of fetch: *An array of* ***complete article objects*** *with id, title, and body.*

This pattern is critical for bridging **storage optimization** (keys as IDs)

with

UI needs (arrays of full objects)

```
[
  {
    id: "art1",
    title: "Introduction to JavaScript",
    body: "JavaScript powers..."
  },
  {
    id: "art2",
    title: "CSS Flexbox Guide",
    body: "Flexbox simplifies..."
  }
]
```

```
export async function fetchArticles() {
  // In storage the ids are separated from the data, but in this function
  // we are going to combine the id and the data together.
  return Object.entries(articles).map(([id, data]) => ({ id, ...data }))
}
```

```
articles = {
   id1: { title: ..., body: ..., date: ... },
   id2: { title: ..., body: ..., date: ... },
}
```

```
[
   [id1, { title: ..., body: ..., date: ... }],
   [id2, { title: ..., body: ..., date: ... }]
]
```

```
const articles = {
  "art1": {
    title: "Introduction to JavaScript",
    body: "JavaScript powers interactive web apps..."
  },
  "art2": {
    title: "CSS Flexbox Guide",
    body: "Flexbox simplifies layout design..."
  }
};
```

```
[
  ["art1", { title: "Introduction to JavaScript", body: "JavaScript powers..." }],
  ["art2", { title: "CSS Flexbox Guide", body: "Flexbox simplifies..." }]
]
```

Why Spread (...data)?
Cleanly merges nested properties (title, body) without manual assignment.

```
[
  { id: id1, title: ..., body: ..., date: ... },
  { id: id2, title: ..., body: ..., date: ... }
]
```

```
{
  id: "art1",              // From destructured `id`
  title: "Introduction to JavaScript",   // From spread `...data`
  body: "JavaScript powers..."           // From spread `...data`
}
```

Recall we said that there were:

- ✓ No HTTP call
- ✓ No server
- ✓ No database
- ✓ No fetch()
- ✓ No axios()
- ✓ No network delay

- ✓ Acts as a gatekeeper between app and database
- ✓ Provides clear interface for CRUD operations
- ✓ Hides database implementation details
- ✓ Centralizes data access logic

Replacing/modifying *articleService.js*

Back to Fire store discussion

App.jsx

```
import { fetchArticles, createArticle } from "../services/articleService"
```

Replace the entire file:

src/services/articleService.js

fireBaseConfig.js

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

src/services/articleService.js

1

```
import { db } from "../fireBaseConfig"
import {
  collection,
  query,
  getDocs,
  addDoc,
  orderBy,
  limit,
  Timestamp,
} from "firebase/firestore"
```

CRUD Operations Table:

Operation	Service Function	Database Equivalent
Create	createArticle()	addDoc()
Read	fetchArticles()	getDocs()
Update	(not shown)	updateDoc()
Delete	(not shown)	deleteDoc()

More to be done!

CRUD Operation	Service Function	Firestore Method	Status	Notes
Create	createArticle()	addDoc()	✅ Implemented	Creates new articles with auto-generated ID and timestamp
Read (All)	fetchArticles()	getDocs() + query	✅ Implemented	Gets first 20 articles (needs pagination)
Read (Single)	❌ getArticle(id)	getDoc()	❌ To be added	Needed for fetching a single article by ID
Update	❌ updateArticle(id, data)	updateDoc()	❌ To be added	Should update specific fields and add updatedAt timesta mp
Delete	❌ deleteArticle(id)	deleteDoc()	❌ To be added	Should delete document and return ID for confirmation

More to be done!

- *Show the photo of the logged in user*
- *Display error messages from Firebase*
- *??*
- *??*

fireBaseConfig.js

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

src/services/articleService.js

1

```
import { db } from "../fireBaseConfig"
import {
  collection,
  query,
  getDocs,
  addDoc,
  orderBy,
  limit,
  Timestamp,
} from "firebase/firestore"
```

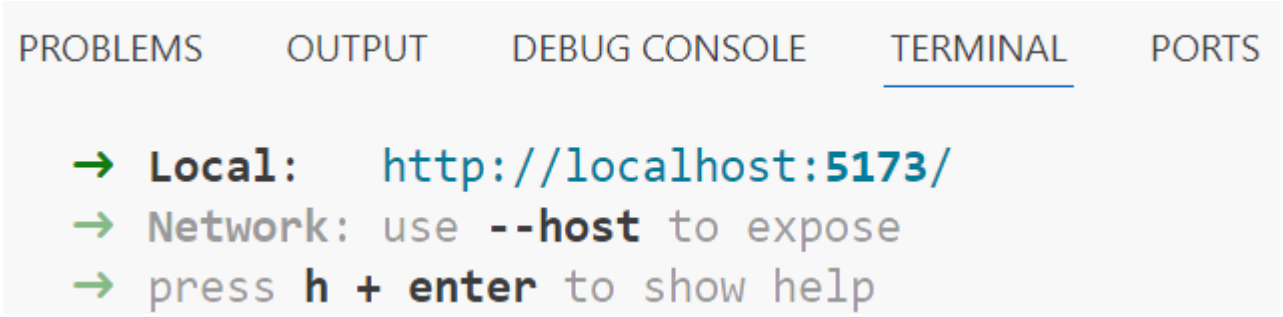
2

```
export async function createArticle({ title, body }) {
  const data = { title, body, date: Timestamp.now() }
  const docRef = await addDoc(collection(db, "articles"), data)
  return { id: docRef.id, ...data }
}
```

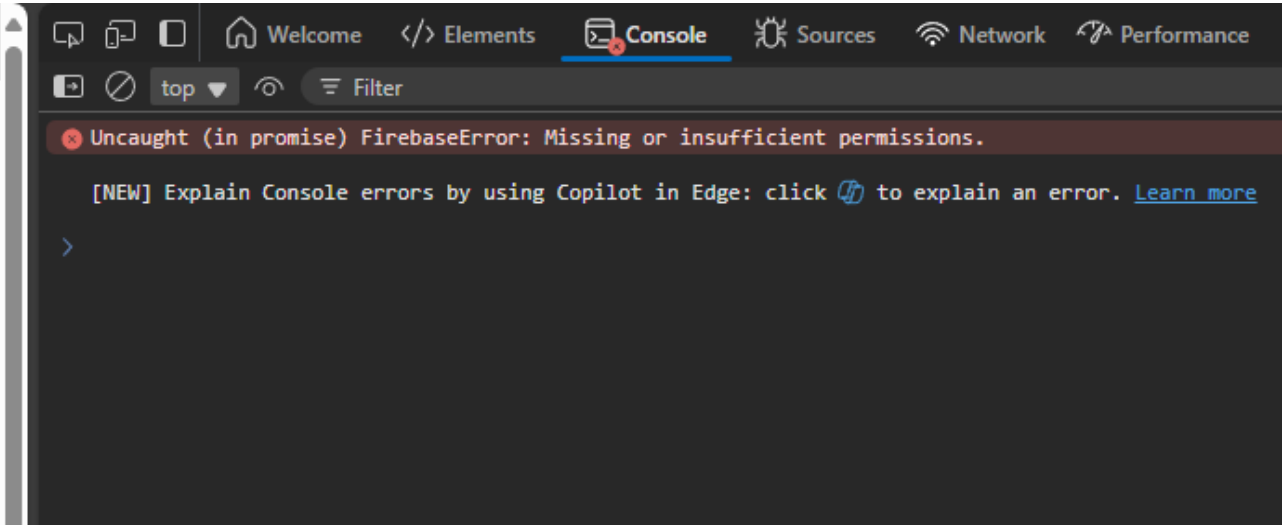
3

```
// NOT FINISHED: This only gets the first 20 articles. In a real app,
// you would implement pagination.
export async function fetchArticles() {
  const snapshot = await getDocs(
    query(collection(db, "articles"), orderBy("date", "desc"), limit(20))
  )
  return snapshot.docs.map((doc) => ({
    id: doc.id,
    ...doc.data(),
  }))
}
```

4



Your console will have an error saying "Insufficient Permissions" so we need to go to our Firebase Firestore console, to the Rules tab. Edit your rules to be:



So, we need to go to our Firebase Firestore console, to the Rules tab. Edit your rules to be:

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

✓ *substituting your Google user id for the asterisk sequence.* *This way, only you can write to the Firestore, but anyone can read as long as they are logged in.*

(You could find your uid on the Auth part of the console if you enabled Google sign-in and you added yourself as the user.)

✓ *Fun test:* *Log out and log in as a different user, and make sure you cannot write!*

Commit

Deploy

More Security!

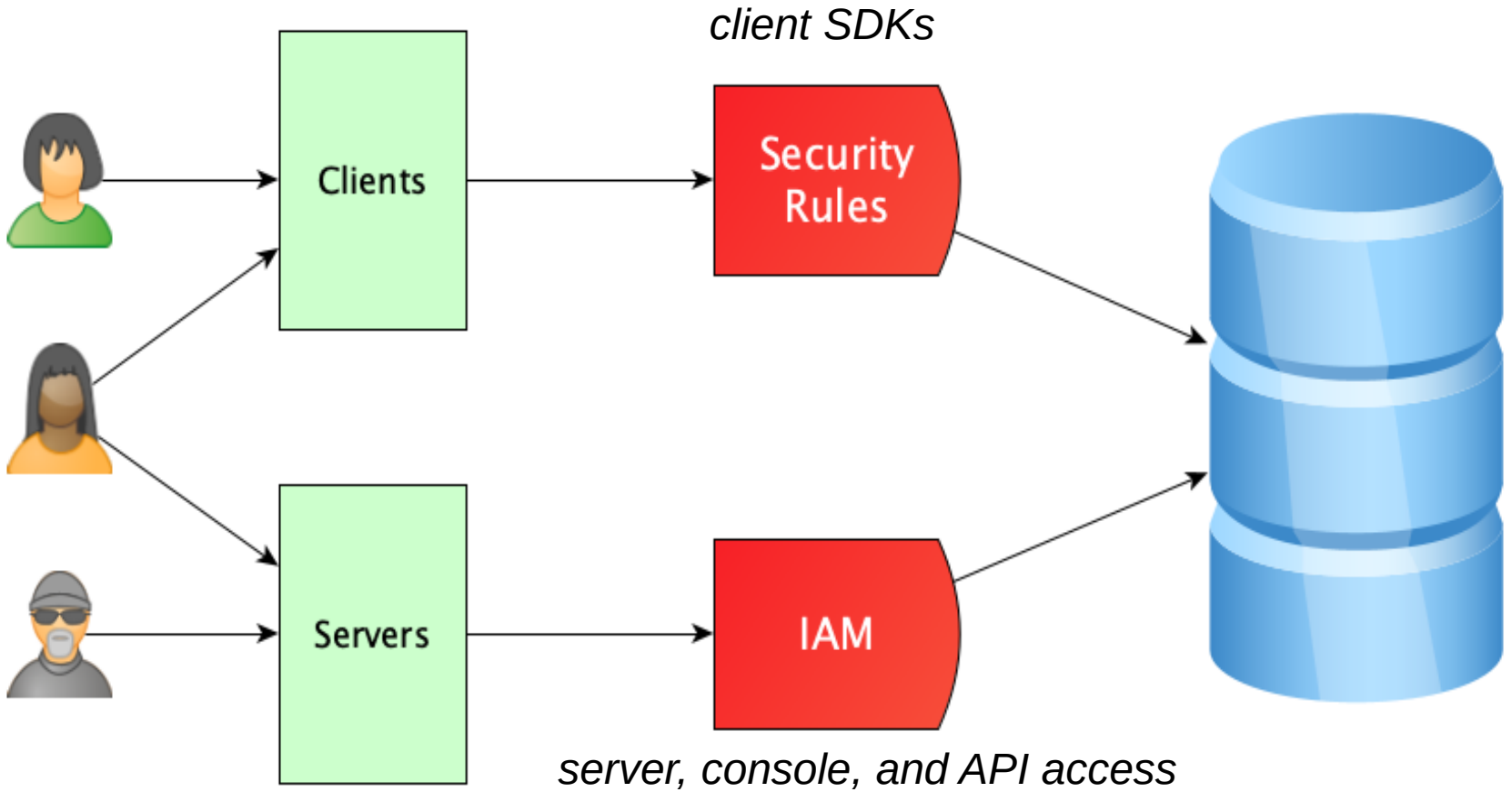
Make improvements

App.jsx

Recall Step 5: Firestore

Let's talk about some authentication stuff first

Firebase is part of the Google Cloud, so if you are connecting to Firebase services from a server, or the console, your authentication and authorization will be configured through Google Cloud's **Identity and Access Management** IAM.



But what about that SDK for the web client?, How can a web client talk to...a database 😳 ? The answer is: **security rules**. You configure rules that will examine all requests from a client—the same kinds of rules you would put in a server that your write yourself.

The configured security rules apply only to client SDKs; server, console, and API access use IAM instead.

- **Client-side:** The user enters credentials (email/password, OTP, etc.).
- **Server-side:** Credentials are **verified** against a DB or identity provider (e.g., Firebase, OAuth).

Always happens on the server side.

Once authenticated, the server checks permissions & roles to control what resources the user can access.

Authentication

Validates the user is who they say they are

(identity)

Authorization

Determines which permissions a user has

(access)

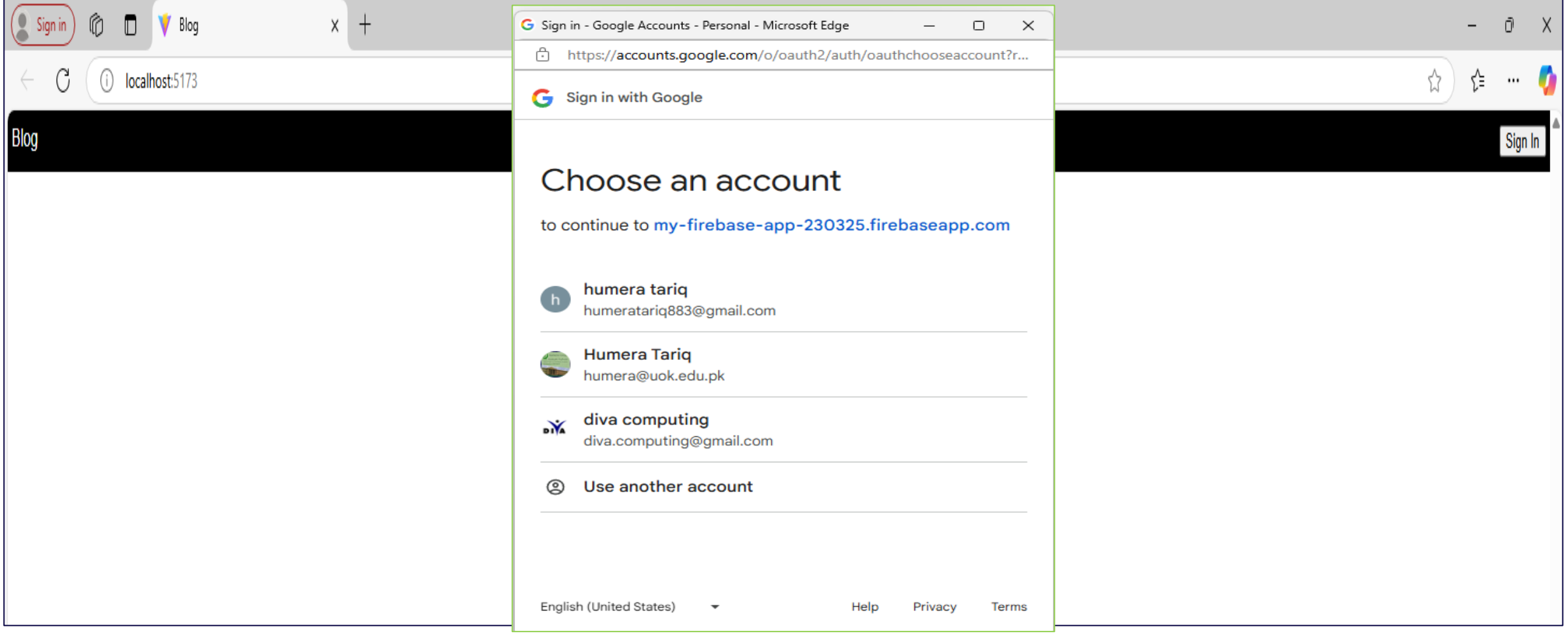
Example: A logged-in user (authenticated) may not have access to the admin dashboard (authorization check fails).

Authentication factors can be:

Something you KNOW: password, passphrase, PIN

Something you HAVE: phone, physical key (USB), email address

Something you ARE: face/voice/fingerprint/retina recognition


Sign in
Blog
localhost:5173
Blog
Sign In
Sign in - Google Accounts - Personal - Microsoft Edge
https://accounts.google.com/o/oauth2/auth/oauthchooseaccount?r...
Sign in with Google
Choose an account
to continue to my-firebase-app-230325.firebaseapp.com
humera tariq
humeratariq883@gmail.com
Humera Tariq
humera@uok.edu.pk
diva computing
diva.computing@gmail.com
Use another account
English (United States)
Help
Privacy
Terms

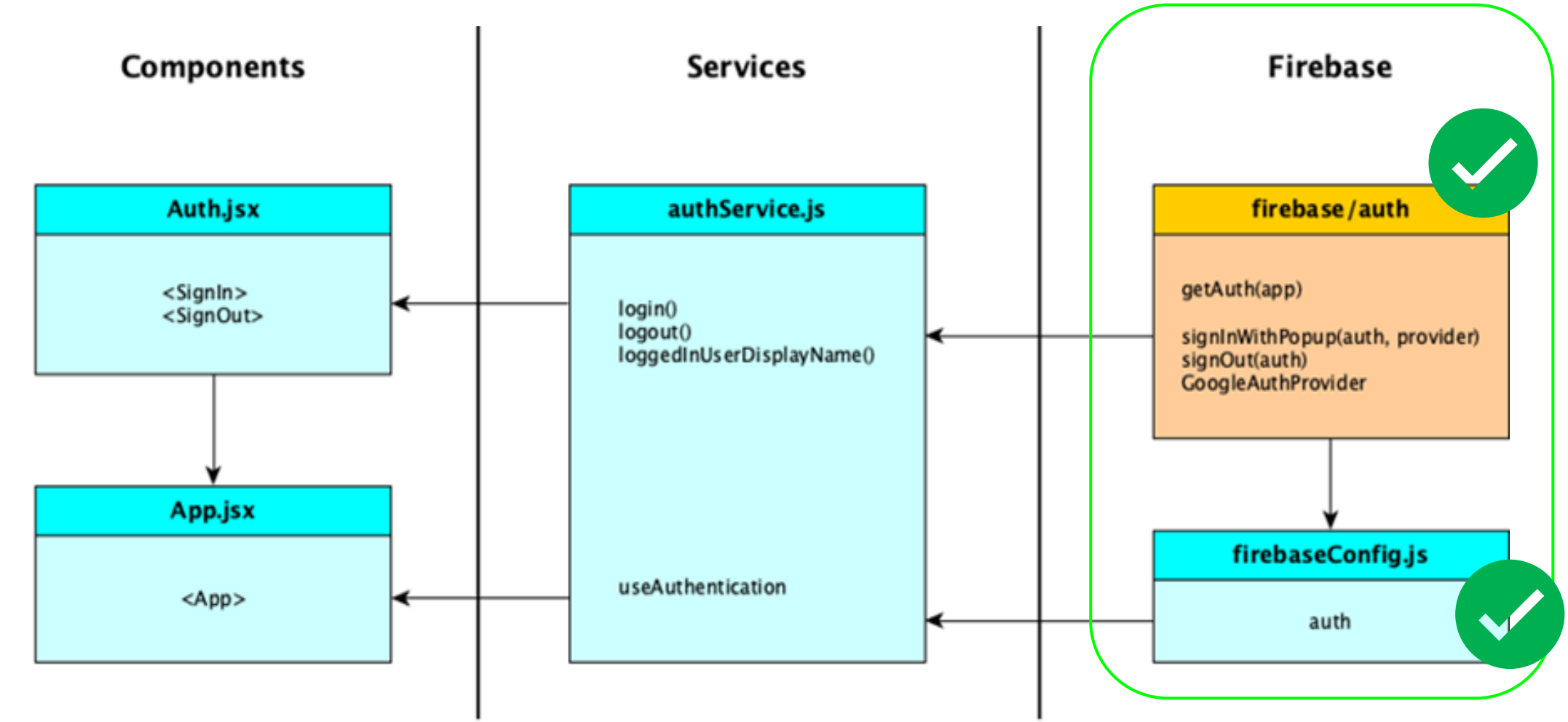

localhost:5173
Blog
New Article
Hello, humera tariq
Sign Out
There's a fair tomorrow
Hello Everyone
No article selected

JS fireBaseConfig.js ●

src > JS fireBaseConfig.js > firebaseConfig

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

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



“How does Google know you’re logged in when you switch pages or reopen the browser?”

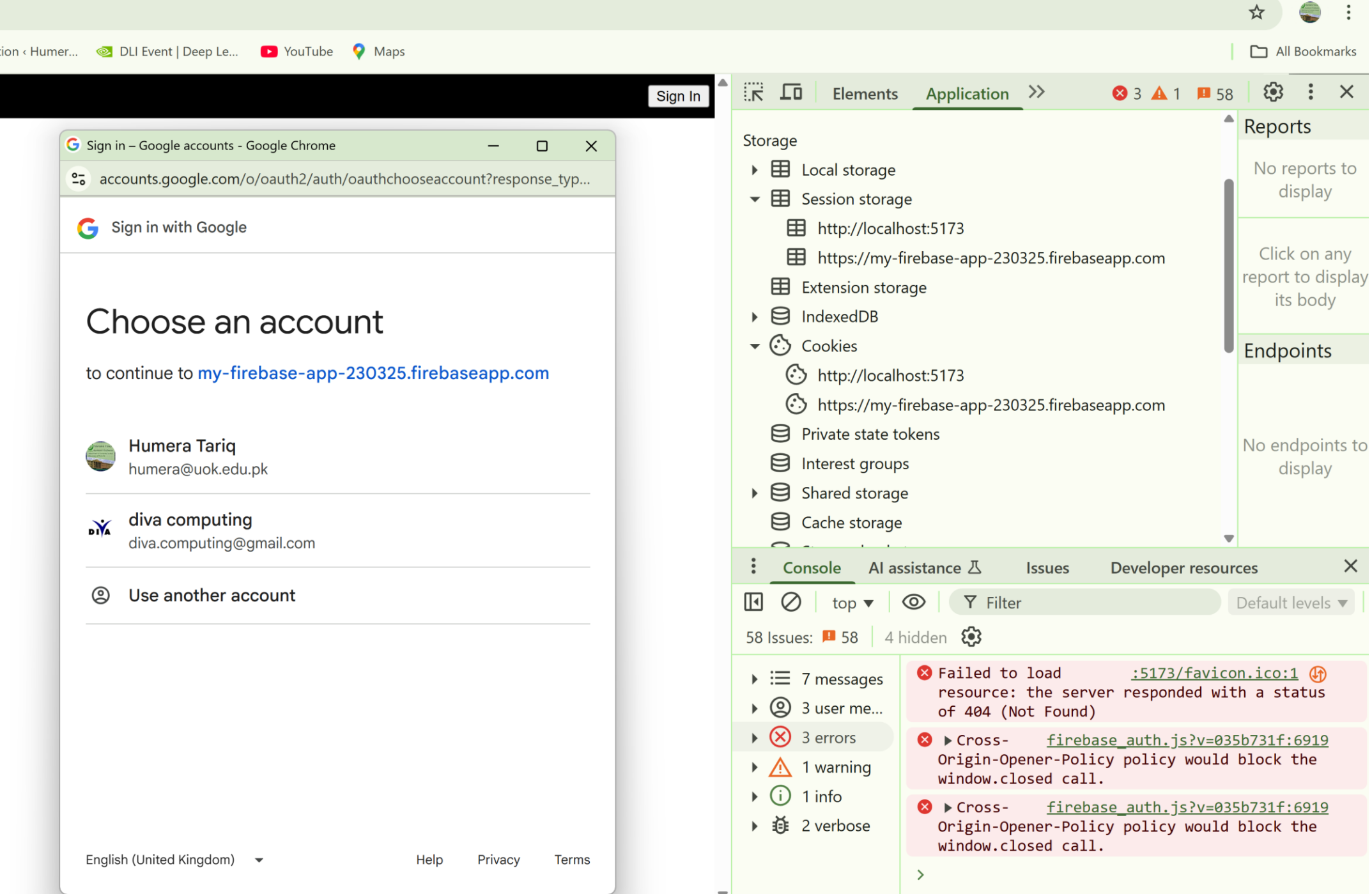
“If you log in to Gmail, how does YouTube recognize you?”

- ✓ Browser communicates with **Google's authentication server** and uses _____ to authenticate user identity.
- ✓ _____ saves the token given by Google .
- ✓ You visit a new page → Browser sends the token with the request
- ✓ Google checks the token → Confirms you're logged in
- ✓ You stay logged in!

The token is like a VIP pass for a concert . Every time you enter, security (Google) checks your pass instead of asking your name again.

The web is built on the _____ policy: a security feature that _____ (Hint: restrict/allow) how documents and scripts can interact with resources from another origin. This principle restricts the ways websites can access _____ .

For example, a document from https://a.example is _____ Hint: prevented/allowed from accessing data hosted at https://b.example.


ion ‹ Humer...
DLI Event | Deep Le...
YouTube
Maps
All Bookmarks
Sign In
Sign in – Google accounts - Google Chrome
accounts.google.com/o/oauth2/auth/oauthchooseaccount?response_typ...
Sign in with Google
Choose an account
to continue to my-firebase-app-230325.firebaseapp.com
Humera Tariq
humera@uok.edu.pk
diva computing
diva.computing@gmail.com
Use another account
English (United Kingdom)
Help
Privacy
Terms
Elements
Application
3
1
58
Storage
Local storage
Session storage
http://localhost:5173
https://my-firebase-app-230325.firebaseapp.com
Extension storage
IndexedDB
Cookies
http://localhost:5173
https://my-firebase-app-230325.firebaseapp.com
Private state tokens
Interest groups
Shared storage
Cache storage
Reports
No reports to display
Click on any report to display its body
Endpoints
No endpoints to display
Console
AI assistance
Issues
Developer resources
top
Filter
Default levels
58 Issues: 58
4 hidden
7 messages
3 user me...
3 errors
1 warning
1 info
2 verbose
Failed to load resource: the server responded with a status of 404 (Not Found)
:5173/favicon.ico:1
Cross-Origin-Opener-Policy policy would block the window.closed call.
firebase_auth.js?v=035b731f:6919
Cross-Origin-Opener-Policy policy would block the window.closed call.
firebase_auth.js?v=035b731f:6919

http://localhost

https://my-firebase-app-230325.firebaseapp.com

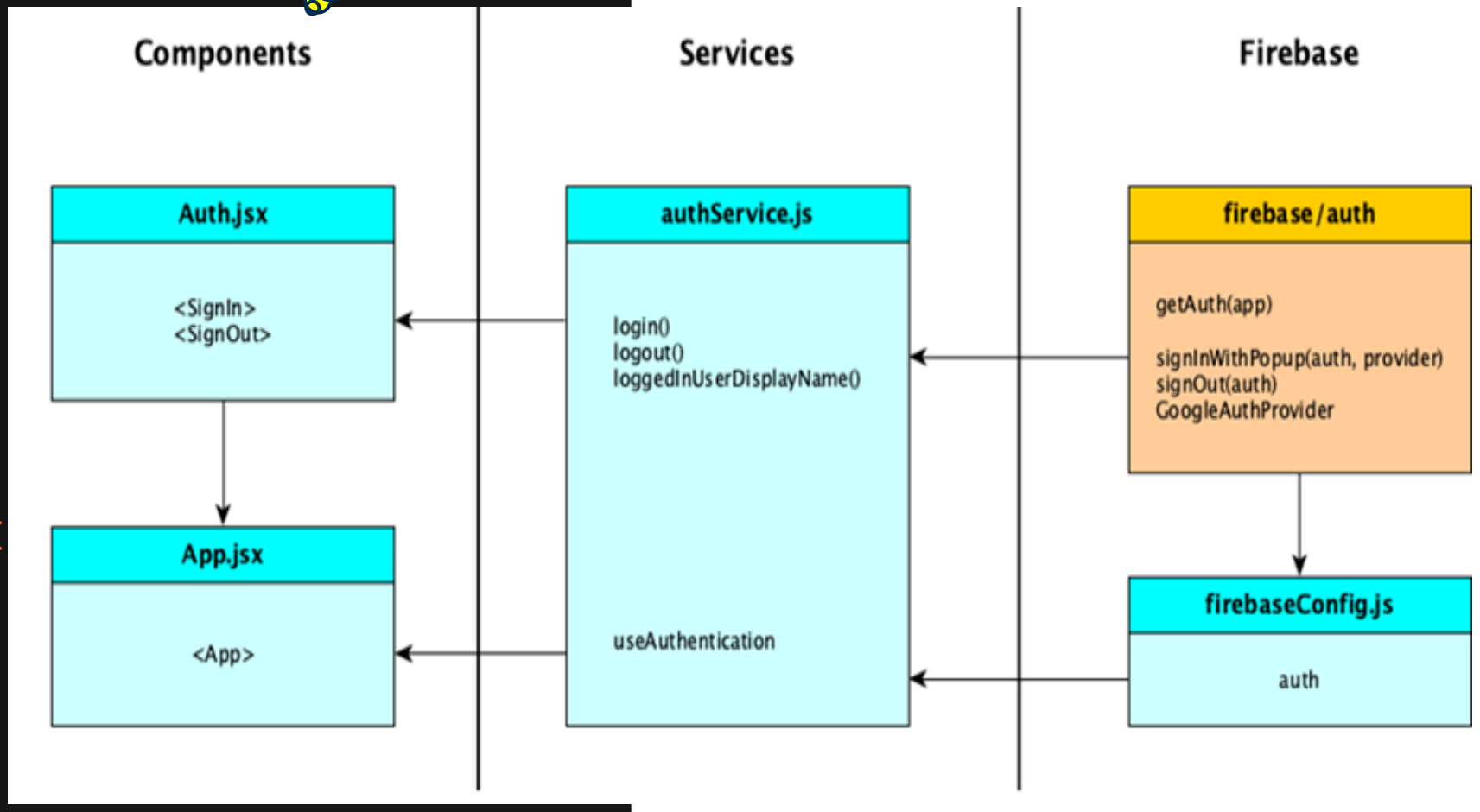
COOP (Cross-Origin Opener Policy) *is a security mechanism that prevents a webpage from interacting with resources from a different origin unless explicitly allowed.*

```jsx
import { useState, useEffect } from "react"
import { signInWithPopup, GoogleAuthProvider, signOut } from "firebase/auth"
import { auth } from "../firebaseConfig"

export function login() {
  return signInWithPopup(auth, new GoogleAuthProvider())
}

export function logout() {
  return signOut(auth)
}

export function loggedInUserDisplayName() {
  return auth.currentUser.displayName
}

export function useAuthentication() {
  const [user, setUser] = useState(null)
  useEffect(() => {
    return auth.onAuthStateChanged((user) => {
      user ? setUser(user) : setUser(null)
    })
  }, [])
  return user
}
```

Recall

src/fireBaseConfig.js
src/services/authService.js

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



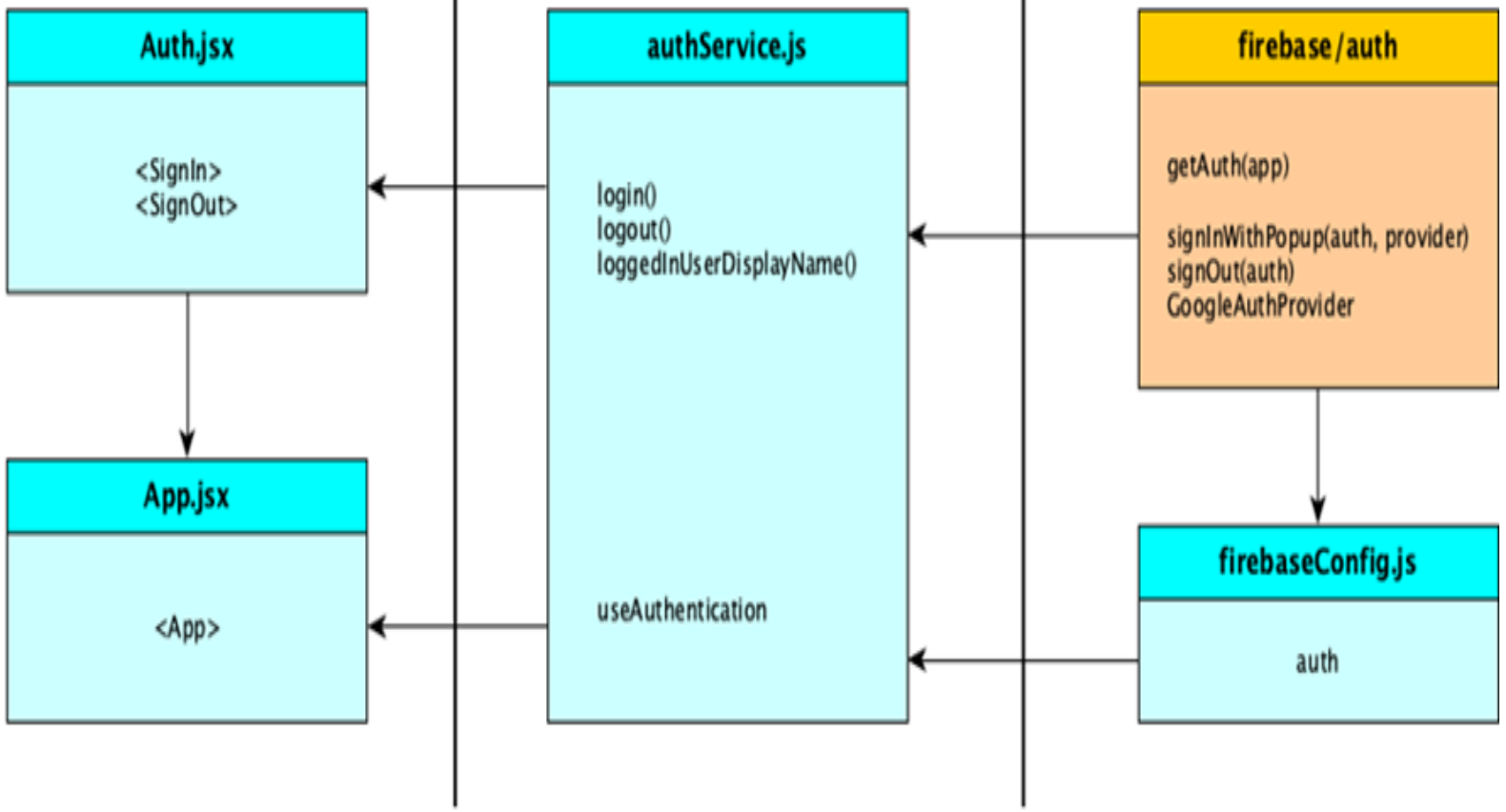
1) *requests authentication via signInWithPopup(…)*

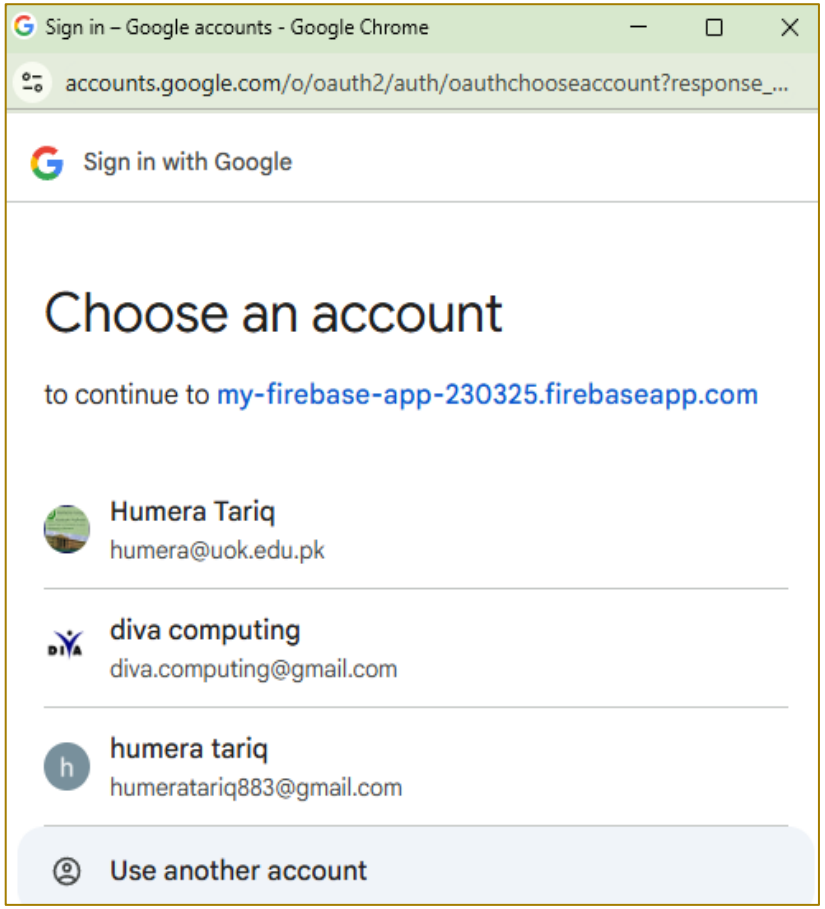
http://localhost → https://my-firebase-app-230325.firebaseapp.com

2) The google sign-in pop up is served from firebase

*3) Firebase processes authentication and **issues an ID token**.*

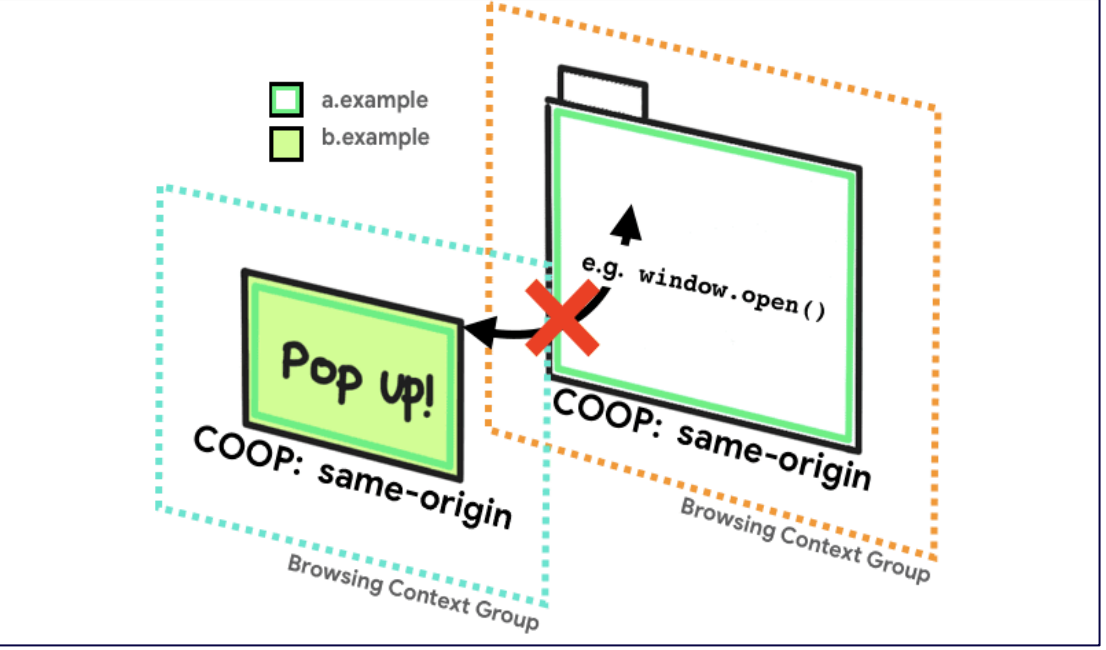
4) Firebase tries to send authentication data back



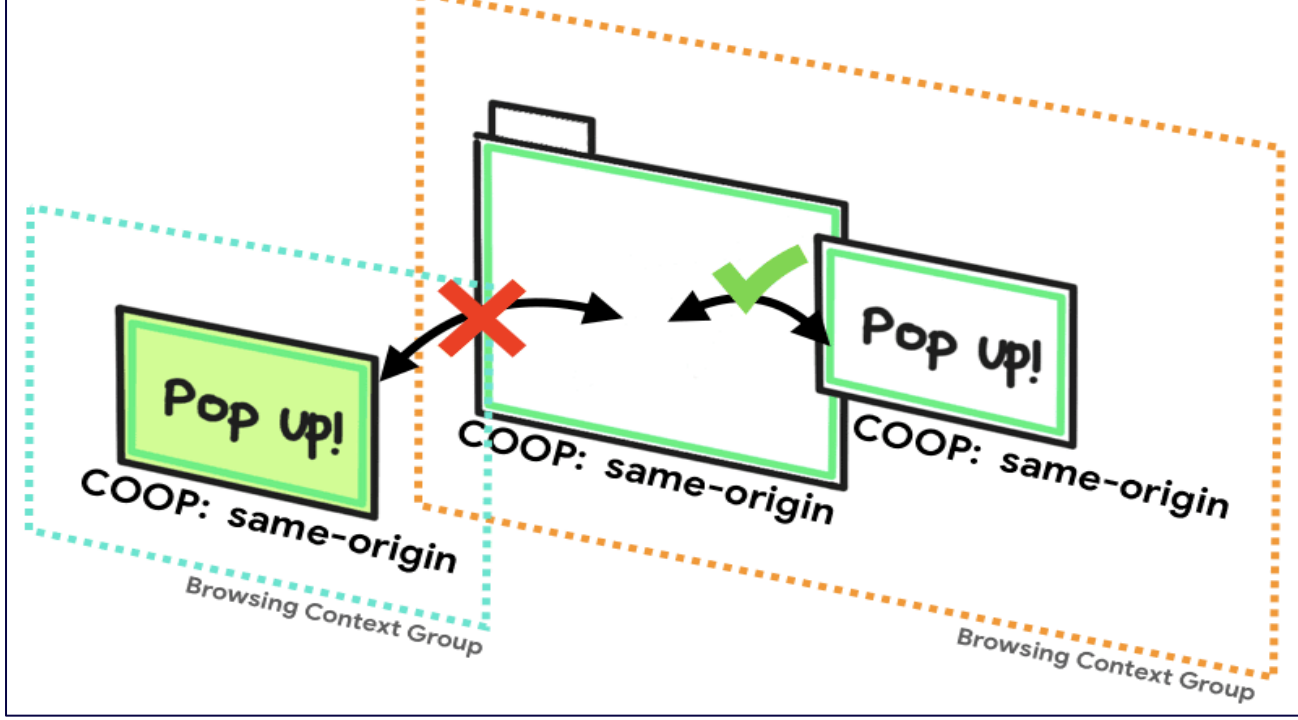


Cross Origin Opener Policy (COOP) allows you to ensure that a top-level window is isolated from other documents by putting them in a different browsing context group, so that they cannot directly interact with the top-level window.

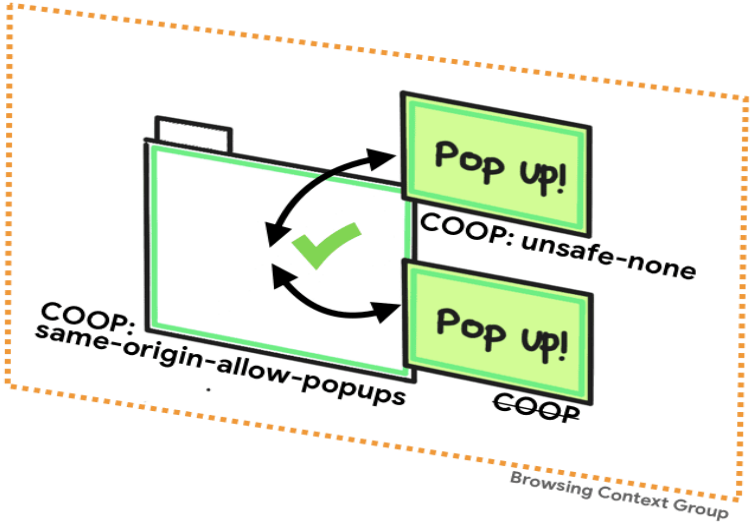
For example, if a document with COOP opens a pop-up, its window.opener property will be null. Also, the .closed property of the opener's reference to it will return true.



1) Documents that are marked **same-origin** can share the same browsing context group with same-origin documents that are also explicitly marked same-origin.



2) same-origin-allow-popups retains references



3) **unsafe-none** is the default and allows the document to be added to its opener's browsing context group unless the opener itself has a COOP of same-origin

The Cross-Origin-Opener-Policy header takes three possible values:

https://web.dev/articles/why-coop-coep

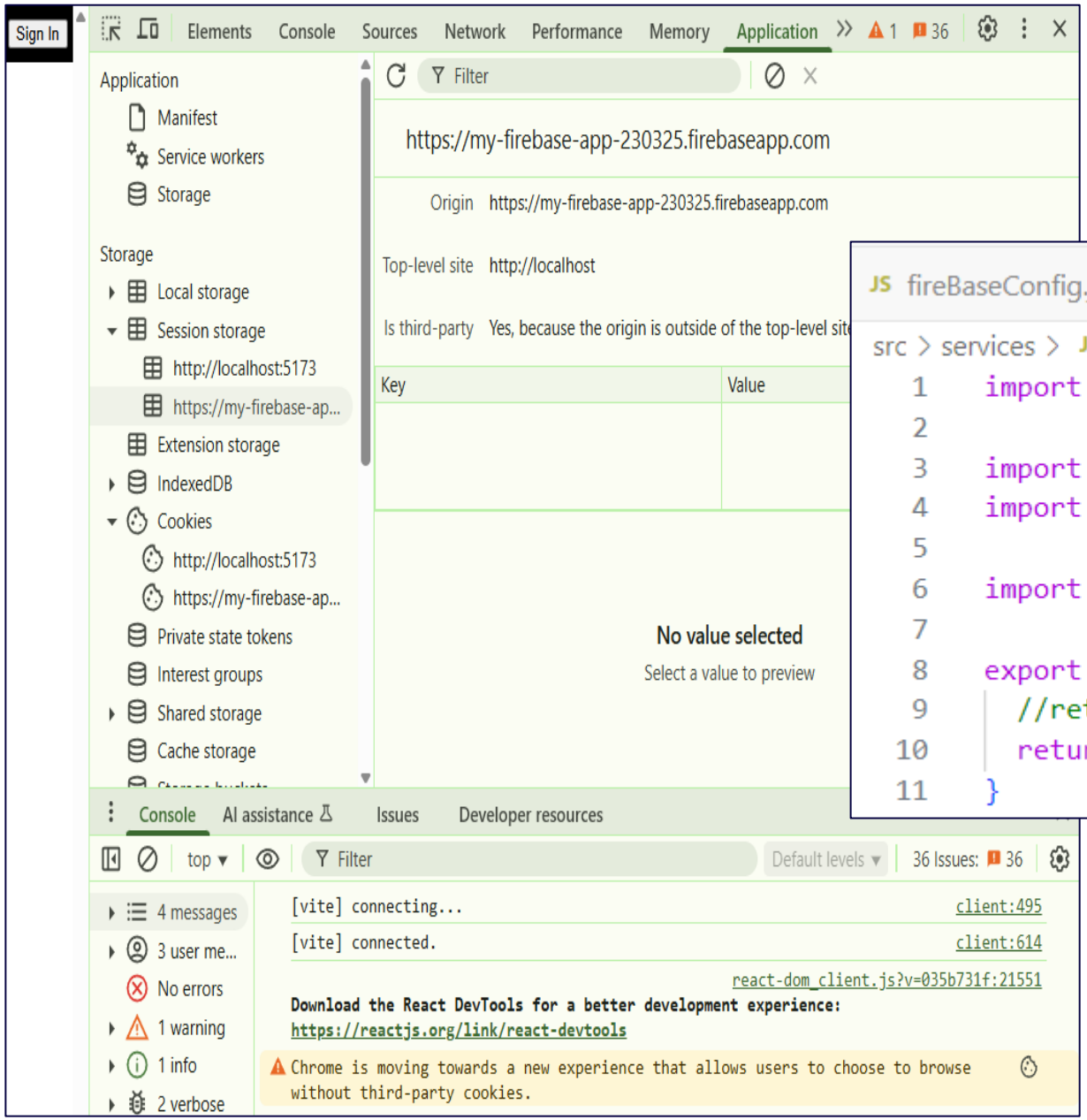
Cross-Origin-Opener-Policy: same-origin

Cross-Origin-Opener-Policy: same-origin-allow pop-ups

Cross-Origin-Opener-Policy: unsafe-none

But in Firebase authentication, since Firebase **controls the popup**, you **cannot directly apply noopener**.

*To **avoid the window.close() block error** **caused by COOP**, you need to **switch from** **signInWithPopup** **to** **signInWithRedirect** in your authentication logic.*



Happy Coding

src/services/authService.js.

fireBaseConfig.js env.txt authService.js

src > services > authService.js > ...

```
1   import { useState, useEffect } from "react"
2
3   import { signInWithPopup, GoogleAuthProvider, signOut } from "firebase/auth"
4   import { signInWithRedirect, getRedirectResult } from "firebase/auth";
5
6   import { auth } from "../fireBaseConfig"
7
8   export function login() {
9     //return signInWithPopup(auth, new GoogleAuthProvider())
10    return signInWithRedirect(auth, new GoogleAuthProvider())
11  }
```

```
import { useState, useEffect } from "react"
import { signInWithPopup, GoogleAuthProvider, signOut } from "firebase/auth"
import { auth } from "../firebaseConfig"

export function login() {
  return signInWithPopup(auth, new GoogleAuthProvider())
}

export function logout() {
  return signOut(auth)
}

export function loggedInUserDisplayName() {
  return auth.currentUser.displayName
}

export function useAuthentication() {
  const [user, setUser] = useState(null)
  useEffect(() => {
    return auth.onAuthStateChanged((user) => {
      user ? setUser(user) : setUser(null)
    })
  }, [])
  return user
}
```

_____ file provide very simple services to the and hide the complexities of _____ .

It will also contain a custom hook to_____ components when the _____state changes.

Call pattern	Code pattern	Execution pattern
No second argument	useEffect(() => {//perform effect});	Run after every render.
Empty array as second argument	useEffect(() => {//perform effect}, [ ]);	Run once, when the component mounts.
Dependency array as second argument	useEffect(() => {//perform effect// that uses dep1 and dep2}, [dep1, dep2]);	Run whenever a value in the dependency array changes.
Return a function	useEffect(() => {//perform effect return () => {/* clean-up */};}, [dep1, dep2]);	React will run the clean-up function when the component unmounts and before re-running the effect.

SKETCHNOTES

useEffect() Illustrated

useEffect is a React hook that helps us "decouple" the side effects from the react render cycle by only triggering them when certain conditions are met

Here's how that works:

no dependency array:
effect() will run on every render

```
useEffect(() => {
    effect()
});
```
1

```
useEffect(() => {
    effect()
}, []);
```
2

empty dependency array:
effect() will run only on first render

non-empty dependency array
effect() will run when any of the deps changes

```
useEffect(() => {
    effect()
}, [dep]);
```
3

part 1/2

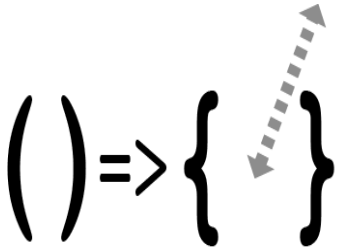
@kokaneka

sketchnotes

https://livebook.manning.com/book/react-hooks-in-action/chapter-4/v-6/76

```
useEffect( () => {

  // perform a side effect

}, [ ] );
```

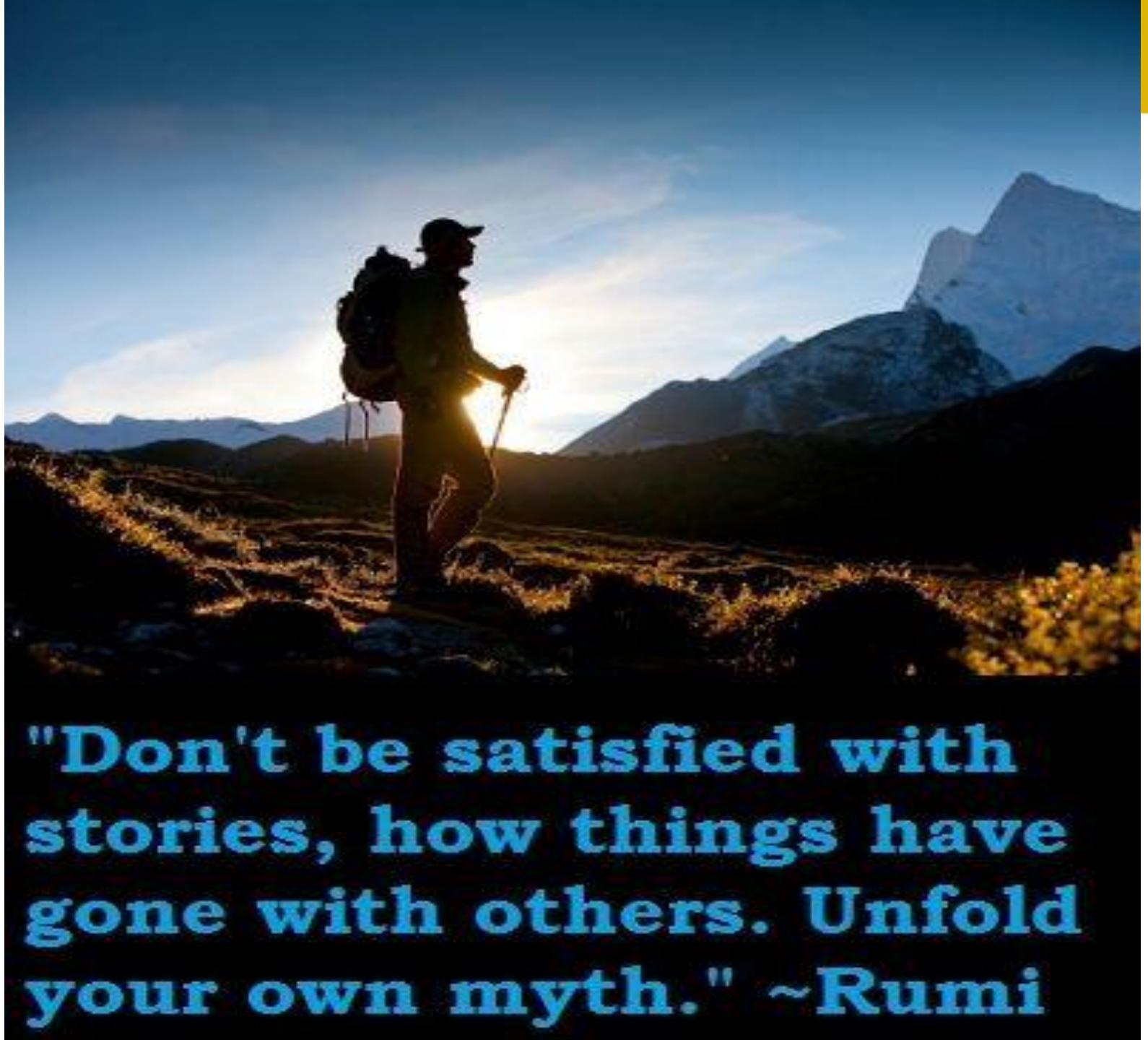
effect function:
We only want to run this function once, when the component mounts



dependency list:
An empty list causes the effect to run once, when the component first mounts

```
export function useAuthentication() {

const [user, setUser] = useState(null)

  useEffect( () => {
    return auth.onAuthStateChanged(   ( _____ ) => {
  _____ ?  _____   : _____ } )
  }, [])

return user
}
```

UNIVERSITY OF
KARACHI



UNIVERSITY OF KARACHI

Department of Compute Science (UBIT Building), Karachi, Pakistan.

1200 Acres (5.2 Km sq.)
53 Departments
19 Institutes
25000 Students

UNIVERSITY OF KARACHI

My Homeland Pakistan