BUILT-IN REACT HOOKS + NGILO APP

I. INTRODUCTION TO HOOKS

React **Hooks** are functions that let you use React features like **state**, **lifecycle**, and **context** inside **functional components**.

Instead of using class components, Hooks make it easier to write cleaner, modular code.

Rules of Hooks:

1. Only call Hooks **at the top level** (not inside loops, conditions, or nested functions).
2. Only call Hooks **inside React function components** or **custom Hooks**.

I. STATE HOOKS

1. useState()

- Allows a component to "remember" a piece of data (like text input or toggle value).
- Updating the state re-renders the component.

Example from Ngilo App – *GenerateLink.jsx*

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

function GenerateLink() {
 const [link, setLink] = useState("");

 useEffect(() => {
  const user =
JSON.parse(localStorage.getItem("user"));
  if (user && user.userId) {
   setLink(`https://ngilo.app/send/${user.userId}`);
  } else {
   setLink("");
  }
 }, []);

 return (
  <div>
   <h2>Your Unique Link</h2>
   {link ? (
    <>
     <p>{link}</p>
     <button onClick={() => {
      navigator.clipboard.writeText(link);
      alert('Link copied!');
     }}>Copy Link</button>
    </>
   ) : (
    <p>Loading your link...</p>
   )}
  </div>
 );
}
export default GenerateLink;
```

📌 **useState stores:** the generated link.

📌 **useEffect loads:** user data from localStorage when the component mounts.

2. useReducer()

- Used when state logic becomes complex (like handling multiple values or actions).

Syntax:

Syntax:

```
js

const [state, dispatch] = useReducer(reducerFunction, initialState);
```

Example

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

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      return state;
  }
}

function CounterReducer() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });
  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>+</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
    </div>
  );
}
```

III. CONTEXT HOOKS

1. useContext()

Allows a component to use shared data without passing props manually.

Syntax:

```
js

const value = useContext(MyContext);
```

Example:

```
js

import React, { createContext, useContext } from 'react';

const ThemeContext = createContext('light');

function Button() {
  const theme = useContext(ThemeContext);
  return <button className={theme}>I am a {theme} button</button>;
}

function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Button />
    </ThemeContext.Provider>
  );
}
```

IV. REF HOOKS

1. useRef()

Stores a value that doesn't cause re-render when changed — commonly used for DOM elements or timers.

Syntax:

```
js

const myRef = useRef(initialValue);
```

Example:

```
js

import React, { useRef } from 'react';

function InputFocus() {
  const inputRef = useRef(null);

  const handleFocus = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef} placeholder="Click button to focus me" />
      <button onClick={handleFocus}>Focus</button>
    </div>
  );
}
```

In Ngilo, this could be used for:

- Auto-focusing on input fields.
- Scrolling message box to the newest message.

V. EFFECT HOOKS

1. `useEffect()`

Performs side effects like fetching data, accessing localStorage, or subscribing to events.

Syntax:

```
js

useEffect(() => {
  // effect logic
  return () => { /* cleanup */ };
}, [dependencies]);
```

Example 1 — Ngilo's Home.jsx (Dynamic Greeting)

```js
useEffect(() => {
  const hour = new Date().getHours();
  setGreeting(
    hour < 12 ? "Good Morning" : hour < 18 ? "Good Afternoon" : "Good Evening"
  );

  const stored = JSON.parse(localStorage.getItem("user"));
  if (stored?.username) {
    setUsername(stored.username);
  } else {
    navigate("/");
  }
}, [navigate]);
```

Purpose:

- Runs only on load.
- Displays a greeting and verifies login.

Example 2 — Ngilo's YourInbox.jsx (Fetching Messages)

```js
useEffect(() => {
  const fetchMessages = async () => {
    try {
      const res = await axios.get(`${API_BASE}/api/messages/${user.userId}`
      setMessages(res.data);
    } catch (err) {
      console.error("Failed to fetch messages:", err);
    } finally {
      setLoading(false);
    }
  };

  if (user?.userId) fetchMessages();
}, [user]);
```

Purpose:
Fetches user's messages once when component mounts.

Example 3 — Ngilo's GenerateLink.jsx (Loading Link)

```js
useEffect(() => {
  const user = JSON.parse(localStorage.getItem("user"));
  if (user && user.userId) {
    setLink(`https://ngilo.app/send/${user.userId}`);
  }
}, []);
```

VI. PERFORMANCE HOOKS

1. useMemo()

Caches a computed value so it's not recalculated every render.

Syntax:

```js
const result = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

Example:

```js
const visibleTodos = useMemo(() => filterTodos(todos, tab), [todos, tab]);
```

You could apply this in Ngilo to optimize **filtered message lists**.

2. useCallback()

Caches function definitions so React doesn't re-create them on each render.

Syntax:

```js
const memoizedFn = useCallback(() => doSomething(), [deps]);
```

Example:

```js
const handleDelete = useCallback((id) => {
  setMessages(messages.filter(msg => msg.id !== id));
}, [messages]);
```

VIII. Other Hooks

1. useLayoutEffect()

Purpose:

Runs synchronously after all DOM mutations but before the browser paints.Use it to measure DOM elements or perform layout calculations before the screen updates.

Syntax:

```js
useLayoutEffect(() => {
  // Code runs before browser repaint
  return () => {
    // Cleanup
  };
}, [dependencies]);
```

Example:

```
import React, { useLayoutEffect, useRef, useState } from 'react';

function BoxSize() {
  const boxRef = useRef();
  const [size, setSize] = useState(0);

  useLayoutEffect(() => {
    setSize(boxRef.current.offsetWidth);
  }, []);

  return (
    <div>
      <div ref={boxRef} style={{ width: '200px', height: '100px', background: 'skyblue' }}>
        Box width: {size}px
      </div>
    </div>
  );
}
```

2. useInsertionEffect()

Purpose:

Runs **before any DOM mutations** — used mainly by CSS-in-JS libraries to **inject dynamic styles**.

Syntax:

```
js

useInsertionEffect(() => {
  // Insert CSS before DOM updates
}, [dependencies]);
```

Example:

```
js

import React, { useInsertionEffect } from 'react';

function DynamicStyle() {
  useInsertionEffect(() => {
    const style = document.createElement('style');
    style.innerHTML = `body { background-color: lavender; }`;
    document.head.appendChild(style);

    return () => {
      document.head.removeChild(style);
    };
  }, []);

  return <p>Dynamic background applied!</p>;
}
```

3. useTransition()

Purpose:

Marks a state update as **non-blocking**, allowing smoother UI transitions for slow updates.

Syntax:

```
js

const [isPending, startTransition] = useTransition();
```

Example:

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

function FilterList() {
  const [input, setInput] = useState('');
  const [list, setList] = useState([]);
  const [isPending, startTransition] = useTransition();

  const handleChange = (e) => {
    const value = e.target.value;
    setInput(value);

    startTransition(() => {
      const newList = Array.from({ length: 20000 }, (_, i) => value + i);
      setList(newList);
    });
  };

  return (
    <div>
      <input type="text" value={input} onChange={handleChange} />
      {isPending && <p>Loading...</p>}
      {list.map((item, i) => <div key={i}>{item}</div>)}
    </div>
  );
}
```

4. useDeferredValue()

Purpose:

Defers updating a non-urgent part of the UI to **keep main interactions responsive**.

Syntax:

```
js

const deferredValue = useDeferredValue(value);
```

Example:

```
js

import React, { useState, useDeferredValue } from 'react';

function SearchList() {
  const [query, setQuery] = useState('');
  const deferredQuery = useDeferredValue(query);
  const results = Array.from({ length: 10000 }, (_, i) => `${deferredQuery} ${i}`);

  return (
    <div>
      <input value={query} onChange={(e) => setQuery(e.target.value)} />
      {results.map((r, i) => <p key={i}>{r}</p>)}
    </div>
  );
}
```

5. useId()

Purpose:

Generates **unique, stable IDs** useful for accessibility attributes or dynamic forms.

Syntax:

```js
const id = useId();
```

Example:

```js
import React, { useId } from 'react';

function Form() {
  const nameId = useId();
  const emailId = useId();

  return (
    <form>
      <label htmlFor={nameId}>Name:</label>
      <input id={nameId} type="text" />
      <label htmlFor={emailId}>Email:</label>
      <input id={emailId} type="email" />
    </form>
  );
}
```

6. useDebugValue()

Purpose:

Used inside custom hooks to label the value displayed in React DevTools.

Syntax:

```js
useDebugValue(value, formatFunction);
```

Example:

```js
import { useState, useEffect, useDebugValue } from 'react';

function useOnlineStatus() {
  const [isOnline, setIsOnline] = useState(navigator.onLine);
  useDebugValue(isOnline ? 'Online 🟢' : 'Offline 🔴');

  useEffect(() => {
    const update = () => setIsOnline(navigator.onLine);
    window.addEventListener('online', update);
    window.addEventListener('offline', update);
    return () => {
      window.removeEventListener('online', update);
      window.removeEventListener('offline', update);
    };
  }, []);

  return isOnline;
}
```

7. useSyncExternalStore()

Purpose:

Subscribes to an external store (e.g., Redux or global state) in a **React-safe way**.

Syntax:

```js
const snapshot = useSyncExternalStore(subscribe, getSnapshot);
```

Example:

```js
import { useSyncExternalStore } from 'react';

const store = {
  value: 0,
  listeners: new Set(),
  subscribe(listener) {
    this.listeners.add(listener);
    return () => this.listeners.delete(listener);
  },
  getSnapshot() {
    return this.value;
  },
  increment() {
    this.value++;
    this.listeners.forEach((l) => l());
  }
};
```

```
function Counter() {
  const count = useSyncExternalStore(store.subscribe.bind(store), store.getSnapshot.bind(store));

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => store.increment()}>Add</button>
    </div>
  );
}
```

8. useActionState()

Purpose:

Manages the state of async form actions or server requests.

Syntax:

```
js

const [state, action, isPending] = useActionState(actionFn, initialState);
```

Example:

```
js

import { useActionState } from 'react';

function SubmitForm() {
  const [state, submitForm, pending] = useActionState(async (prev, formData) => {
    await new Promise((r) => setTimeout(r, 1500)); // simulate delay
    return formData.get('name');
  }, null);

  return (
    <form action={submitForm}>
      <input name="name" />
      <button type="submit" disabled={pending}>Submit</button>
      <p>Last submitted: {state}</p>
    </form>
  );
}
```

9. useImperativeHandle()

🧠 **Purpose:**

Customizes the instance value exposed to parent components using ref.

🧩 **Syntax:**

```
js

useImperativeHandle(ref, () => ({
  methodName: () => { ... }
}), [dependencies]);
```

Example:

```
import React, { useImperativeHandle, useRef, forwardRef } from 'react';

const ChildComponent = forwardRef((props, ref) => {
  const inputRef = useRef();

  useImperativeHandle(ref, () => ({
    focusInput: () => inputRef.current.focus(),
    clearInput: () => inputRef.current.value = ''
  }));

  return <input ref={inputRef} placeholder="Type something..." />;
});

function Parent() {
  const childRef = useRef();

  return (
    <div>
      <ChildComponent ref={childRef} />
      <button onClick={() => childRef.current.focusInput()}>Focus</button>
      <button onClick={() => childRef.current.clearInput()}>Clear</button>
    </div>
  );
}
```

IX. BACKEND QUICK REVIEW (Express + SQLite)

This supports your React app.

```
const express = require('express');
const sqlite3 = require('sqlite3');
const bcrypt = require('bcrypt');
const { v4: uuidv4 } = require('uuid');
const cors = require('cors');

const app = express();
const PORT = 3001;

app.use(cors());
app.use(express.json());

// DATABASE CONNECTION
const db = new sqlite3.Database('./test.db', (err) =>
{
  if (err) console.error('DB Error:', err.message);
  else console.log('Connected to test.db');
});

// CREATE TABLES
db.run(`CREATE TABLE IF NOT EXISTS users (
  user_id TEXT PRIMARY KEY,
  username TEXT UNIQUE,
  password TEXT
```

```
)`);

db.run(`CREATE TABLE IF NOT EXISTS messages (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  user_id TEXT,
  message TEXT,
  timestamp DATETIME DEFAULT
CURRENT_TIMESTAMP,
  FOREIGN KEY(user_id) REFERENCES users(user_id)
)`);

// REGISTER USER
app.post('/api/register', async (req, res) => {
  const { username, password } = req.body;
  const hashedPassword = await
bcrypt.hash(password, 10);
  const userId = uuidv4();

  db.run(`INSERT INTO users (user_id, username,
password) VALUES (?, ?, ?)`,
    [userId, username, hashedPassword],
    function (err) {
      if (err) return res.status(400).send('Username
already exists.');
      res.status(200).send({ userId, username });
    });
});

// LOGIN USER
app.post('/api/login', (req, res) => {
  const { username, password } = req.body;
  db.get(`SELECT * FROM users WHERE username =
?`, [username], async (err, user) => {
    if (!user) return res.status(400).send('Invalid
username');
    const match = await bcrypt.compare(password,
user.password);
    if (!match) return res.status(400).send('Invalid
password');
    res.status(200).json({ userId: user.user_id,
username: user.username });
  });
});

// SEND MESSAGE
app.post('/api/send-message', (req, res) => {
  const { recipientId, message } = req.body;
  db.run(`INSERT INTO messages (user_id, message)
VALUES (?, ?)`, [recipientId, message], (err) => {
    if (err) return res.status(500).send('Failed to send
message.');
    res.status(200).send('Message sent
successfully.');
  });
});

// GET MESSAGES
app.get('/api/messages/:userId', (req, res) => {
  const { userId } = req.params;
  db.all(`SELECT message, timestamp FROM
messages WHERE user_id = ? ORDER BY timestamp
DESC`,
    [userId],
    (err, rows) => {
      if (err) return res.status(500).send('Failed to
retrieve messages.');
      res.status(200).json(rows);
    });
});

app.listen(PORT, () => console.log(`Server running
on port ${PORT}`));
```

X. FRONTEND QUICK REVIEW

Overview

The **Ngilo App** frontend is built using **React.js**.
It uses **functional components** with **React Hooks** to handle:

- State management (useState)
- Side effects (useEffect)
- Navigation (useNavigate)
- Animations (framer-motion)
- API requests (axios and fetch)
- Local storage (localStorage)

The app flow includes:

1. **Register** → Create user account
2. **Login** → Authenticate and save user data
3. **Home Dashboard** → Displays greeting, navigation cards
4. **Generate Link** → Create a unique anonymous message link
5. **Message** → Send anonymous message
6. **Your Inbox** → View received messages

🏠 Home.jsx

🧠 Purpose:

Main dashboard shown after login. Displays greeting, username, and options to open different features.

📘 Hooks Used:

- useState → for greeting, username, and popup control
- useEffect → for dynamic greeting + verifying login
- useNavigate → for redirecting user

💻 Code Example:

```jsx
import React, { useEffect, useState } from "react";
import { motion, AnimatePresence } from "framer-motion";
import { useNavigate } from "react-router-dom";
import GenerateLink from "../Components/GenerateLink";
import Message from "../Components/Message";
import YourInbox from "../Components/YourInbox";
import "../App.css";

function Home() {
  const [greeting, setGreeting] = useState("");
  const [username, setUsername] = useState("User");
  const [showComponent, setShowComponent] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    const hour = new Date().getHours();
    setGreeting(
      hour < 12 ? "Good Morning" : hour < 18 ? "Good Afternoon" : "Good Evening"
    );

    const stored = JSON.parse(localStorage.getItem("user"));
    if (stored?.username) setUsername(stored.username);
    else navigate("/");
  }, [navigate]);
```

```jsx
const handleLogout = () => {
  localStorage.removeItem("user");
  navigate("/");
};

return (
  <div className="mg-dashboard">
    <motion.div className="dashboard-shell" initial={{ opacity: 0, y: 20 }}
      animate={{ opacity: 1, y: 0 }} transition={{ duration: 0.7 }}>

      <div className="dashboard-header">
        <h1>{greeting}, <span>{username}</span></h1>
        <p>Welcome to your creative space — what would you like to do?</p>
        <button onClick={handleLogout}>Log Out</button>
      </div>

      <div className="dashboard-grid">
        <div onClick={() => setShowComponent("message")}>Send Message</div>
        <div onClick={() => setShowComponent("generate")}>Generate Link</div>
        <div onClick={() => setShowComponent("inbox")}>Your Inbox</div>
      </div>
```

```jsx
        <AnimatePresence>
          {showComponent && (
            <motion.div className="popup">
              <button onClick={() => setShowComponent(null)}>✖</button>
              {showComponent === "generate" && <GenerateLink />}
              {showComponent === "message" && <Message />}
              {showComponent === "inbox" && <YourInbox />}
            </motion.div>
          )}
        </AnimatePresence>
      </motion.div>
    </div>
  );
}

export default Home;
```

💌 Message.jsx

🧠 Purpose:

Allows users to send anonymous messages using another user's shared link.

📘 Hooks Used:

- useState → manages input and status messages

💻 Code Example:

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

function Message() {
  const [link, setLink] = useState('');
  const [message, setMessage] = useState('');
  const [status, setStatus] = useState('');

  const handleSend = async () => {
    setStatus('');
    try {
      const parts = link.split('/send/');
      const recipientId = parts[1];

      if (!recipientId || !message.trim()) {
        setStatus('Please enter a valid link and message.');
        return;
      }

      const res = await fetch('http://localhost:3001/api/send-message', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ recipientId, message }),
      });
```

```
    const text = await res.text();
    if (!res.ok) return setStatus(`Error: ${text}`);

    setStatus('Message sent anonymously!');
    setMessage('');
    setLink('');
  } catch (error) {
    setStatus('Error sending message.');
    console.error(error);
  }
 };

 return (
  <div>
   <h2>Send Anonymous Message</h2>
   <input
    type="text"
    placeholder="Paste recipient's link"
    value={link}
    onChange={(e) => setLink(e.target.value)}
   />
   <textarea
    placeholder="Your message"
    value={message}
    onChange={(e) => setMessage(e.target.value)}
   />
   <button onClick={handleSend}>Send</button>
   {status && <p>{status}</p>}
  </div>
 );
}

export default Message;
```

🔗 **GenerateLink.jsx**

🧠 **Purpose:**

Creates a unique shareable link for each user so others can send anonymous messages.

📘 **Hooks Used:**

- useState → store the generated link
- useEffect → load the logged-in user from localStorage

📥 **YourInbox.jsx**

🧠 **Purpose:**

Fetches and displays messages received by the logged-in user.
Also allows deleting, archiving, or hiding messages.

📘 **Hooks Used:**

- useState → manage messages, loading, and archived/hidden states
- useEffect → fetch messages on mount

💻 **Code Example:**

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

const YourInbox = () => {
 const [messages, setMessages] = useState([]);
 const [loading, setLoading] = useState(true);
 const user =
JSON.parse(localStorage.getItem("user"));
 const API_BASE = "http://localhost:3001";

 useEffect(() => {
  const fetchMessages = async () => {
   try {
    const res = await
axios.get(`${API_BASE}/api/messages/${user.userId
}`);
    setMessages(res.data);
   } catch (err) {
    console.error("Error fetching messages:", err);
   } finally {
    setLoading(false);
   }
  };

  if (user?.userId) fetchMessages();
 }, [user]);

 const handleDelete = async (id) => {
  try {
   await
axios.delete(`${API_BASE}/api/message/${id}`);
   setMessages(messages.filter((msg) => msg.id
!== id));
  } catch (err) {
   console.error("Error deleting message:", err);
  }
 };

 if (loading) return <p>Loading messages...</p>;

 return (
  <div>
   <h2>Your Inbox</h2>
```

```
    {messages.length === 0 ? (
      <p>No messages yet.</p>
    ) : (
      messages.map((msg) => (
        <div key={msg.id}>
          <p>{msg.message}</p>
          <button onClick={() =>
handleDelete(msg.id)}>Delete</button>
        </div>
      ))
    )}
  </div>
 );
};

export default YourInbox;
```

🔑 **Login.jsx**

🧠 **Purpose:**

Logs in the user by sending credentials to the backend and saving their data in localStorage.

📘 **Hooks Used:**

- useState → handle input and errors
- useNavigate → redirect after login

💻 **Code Example:**

```
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-
dom';

const Login = () => {
 const [username, setUsername] = useState('');
 const [password, setPassword] = useState('');
 const [error, setError] = useState('');
 const navigate = useNavigate();

 const handleLogin = async (e) => {
  e.preventDefault();
  setError('');
  try {
   const response = await
fetch('http://localhost:3001/api/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username, password })
   });
   const data = await response.json();
   if (response.ok) {
    localStorage.setItem('user',
JSON.stringify(data));
    navigate('/home');
   } else setError(data.message || 'Invalid
username or password');
  } catch {
   setError('Server error. Please try again later.');
  }
 };

 return (
  <div>
   <h2>Log In</h2>
   <form onSubmit={handleLogin}>
    <label>Username:</label>
    <input value={username} onChange={(e) =>
setUsername(e.target.value)} />
    <label>Password:</label>
    <input type="password" value={password}
        onChange={(e) =>
setPassword(e.target.value)} />
    <button type="submit">Log In</button>
   </form>
   {error && <p style={{ color: 'red' }}>{error}</p>}
   <Link to="/register">Sign Up</Link>
  </div>
 );
};

export default Login;
```

📝 **Register.jsx**

🧠 **Purpose:**

Registers a new user and displays their assigned userId after successful signup.

📘 **Hooks Used:**

- useState → manage form fields
- axios.post() → send registration request

💻 **Code Example:**

```
import React, { useState } from 'react';
import axios from 'axios';
import { Link } from 'react-router-dom';

const Register = () => {
 const [username, setUsername] = useState('');
 const [password, setPassword] = useState('');

 const handleSignup = async (e) => {
```

```
    e.preventDefault();
    try {
      const res = await
axios.post('http://localhost:3001/api/register', {
username, password });
      alert('Signup successful! Your ID: ' +
res.data.userId);
    } catch (err) {
      alert('Signup failed: ' + err.response.data);
    }
  };

  return (
    <div>
      <h2>Sign Up</h2>
      <form onSubmit={handleSignup}>
        <label>Username:</label>
        <input value={username} onChange={(e) =>
setUsername(e.target.value)} />
        <label>Password:</label>
        <input type="password" value={password}
onChange={(e) => setPassword(e.target.value)} />
        <button type="submit">Sign Up</button>
      </form>
      <Link to="/">Back to Login</Link>
    </div>
  );
};

export default Register;
```

📘 **Summary Table**

File	Hooks Used	Purpose
Home.jsx	useState, useEffect, useNavigate	Dashboard control & logout
GenerateLink.jsx	useState, useEffect	Generate unique link
Message.jsx	useState	Send anonymous message
YourInbox.jsx	useState, useEffect	Fetch and manage inbox
Login.jsx	useState, useNavigate	Handle login flow
Register.jsx	useState	Register new user