Batch: D2 Roll No: 16010122323

Exp No: 4

Title: Write in detail the implementation Process

Objective: To understand the implementation process of the project.

Expected Outcome of Experiment:

Course Outcome	After successful completion of the course students should be able to
CO 4	Implement and test the design as per mentioned specifications.

Books/ Journals/ Websites referred:

1.
2.
3.

Introduction:

Describe the need of this stage in project:

In the To-Do List application project, the implementation process is indispensable for transforming conceptual solutions into tangible features. It's the stage where the app's functionalities, designed to address task overload, are developed and integrated. Through meticulous coding, testing, and iteration, the app's user interface, task management algorithms are refined to ensure usability and effectiveness. Implementation bridges the gap between vision and reality, aligning the app with user needs and preferences. It's essential for delivering a high-quality product that empowers users to prioritize tasks, manage their workload efficiently.

SOMAIYA VIDYAVIHAR UNIVERSITY
K. J. Somaiya College of Engineering, Mumbai -77
(A Constituent College of Somaiya Vidyavihar University)

Title of Mini Project: To-do List

Team Members:
1. Kartik Ambupe- 16010122318
2. Jiya Trivedi 16010122321
3. Vedansh Savla 16010122323

Implementation details :

Eg: API as always-on software sitting on a server, which acts as the interface between the frontend, the web JavaScript app on the browser in this example, and other code which manipulates data and accesses a database.

Access to the API from the JavaScript front end is done through the HTTP protocol. In the case of a web application, the API returns JSON, a specific data format to transfer specific information.

Pushing data to the API will also be usually done with JSON. The back end can be built with Java or Python, built with JavaScript libraries, specifically, NodeJS and the back end services that the web application depends for accessing database.

"Students are supposed to write the backend and frontend all details about the database name and attributes, fields etc."

QuickTask is a web application built using React.js for the frontend, Firebase As the database and the backend. The application is currently deployed on Vercel for the frontend and on Azure as an always-on virtual machine for the backend.

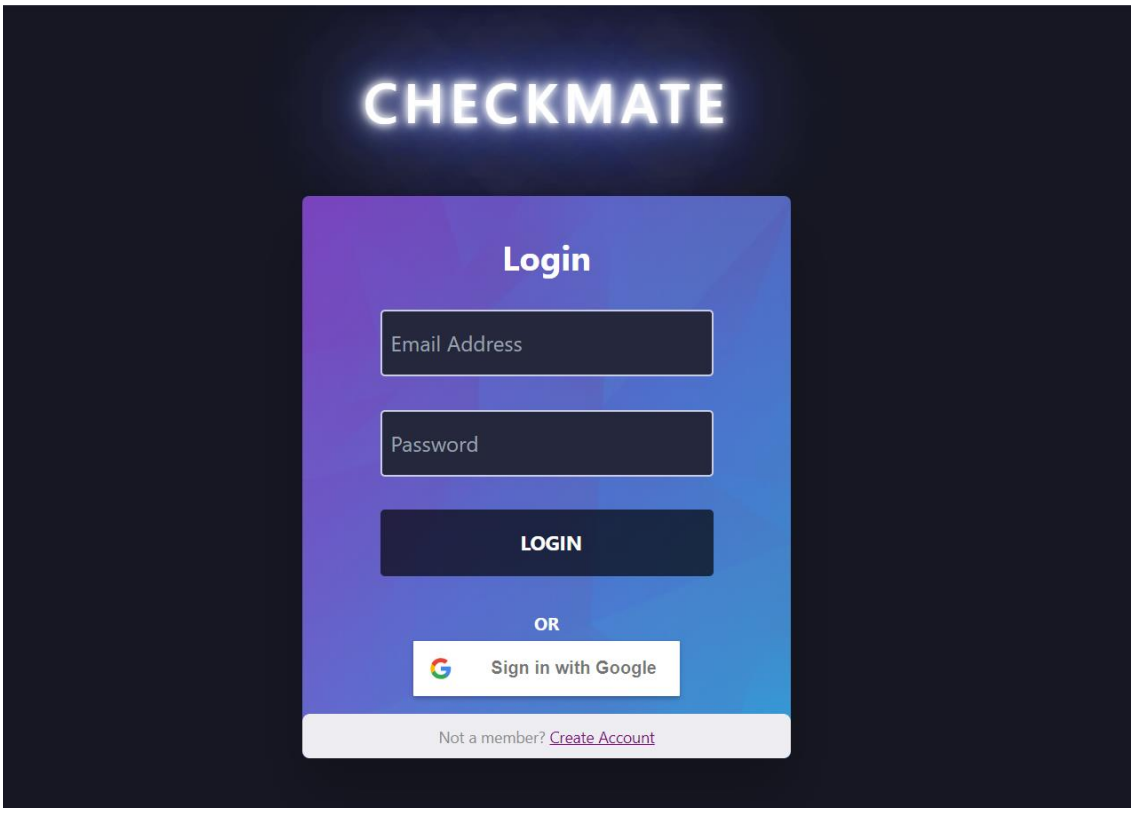
The backend of QuickTask is designed to provide easy OAuth-based login, allowing users to authenticate with their Google account and access their account info. This encourages users to efficiently manage their daily task.

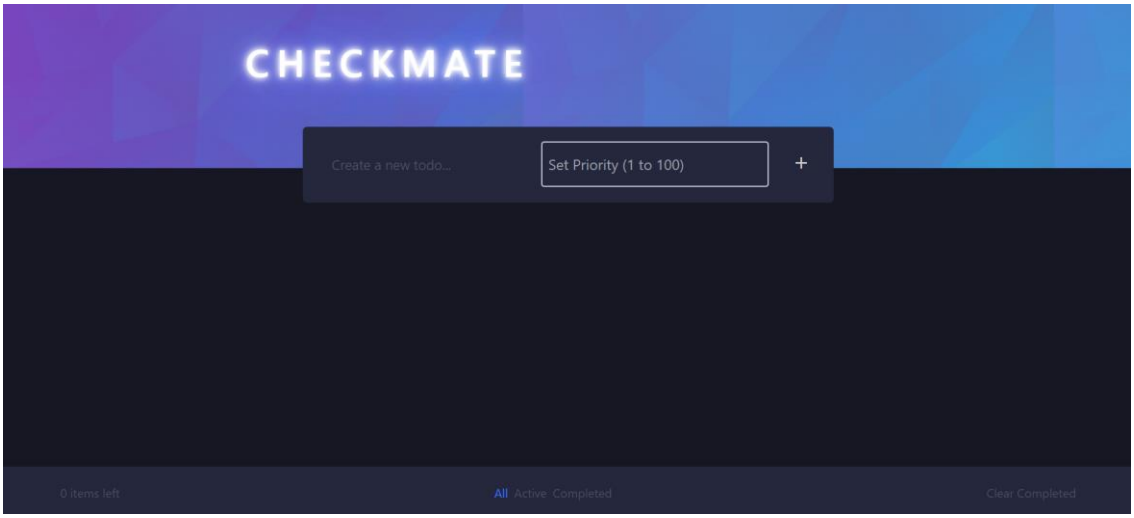
Functions Implemented: ***Describe major and minor functions***

1. Signup with Google/Email-Password for New users.
2. Login for registered users
3. Add Tasks
4. Edit Tasks
5. Delete Tasks
6. Check Tasks
7. Set priority
8. Real-time synchronization of task data across devices using Firebase's real-time database
9. Instantaneous updates when tasks are added, edited, or completed by any user.

User interface:

Implementation method: ***Write the way you have implemented different modules and put the screenshots after implementation***





GOOGLE SIGN AND LOGIN LOGIC:

```
const emailRef = useRef()
    const passwordRef = useRef()
    const [error, setError] = useState('')
    const [loading, setLoading] = useState(false)
    const {googleSignIn}= useAuth()
    const navigate = useNavigate()

    async function handleSubmit(e) {
        e.preventDefault();


        try {
            setError('');
            setLoading(true);
            await signInWithEmailAndPassword(auth,
emailRef.current.value, passwordRef.current.value)
            navigate('/todo')
            } catch (err) {
                setError('Login Failed')

            }
            setLoading(false);
    }

    const handleGoogleSignIn = async () => {
      try {
        setError('');
```

```
        setLoading(true);
        await googleSignIn();
        navigate('/todo');
      } catch (error) {
        console.error('Google Sign-In Failed', error);
      }
      setLoading(false);
    };
```

SIGN UP LOGIC:

```
const emailRef = useRef()
    const passwordRef = useRef()
    const passwordConfirmRef = useRef()
    const signup  = useAuth();
    const [error, setError] = useState('')
    const [loading, setLoading] = useState(false)
    const history = useNavigate()
    async function handleSubmit(e) {
        e.preventDefault();
        setError('');

        if (passwordRef.current.value !==
passwordConfirmRef.current.value) {
            setError('Passwords do not match');
            window.alert('Passwords do not match');
            passwordRef.current.value = '';
            passwordConfirmRef.current.value = '';
            return;
        }
        if (passwordRef.current.value.length < 6) {
            setError('Passwords must be 6 or more characters');
            window.alert('Passwords must be 6 or more characters');
            passwordRef.current.value = '';
            passwordConfirmRef.current.value = '';
            return;
        }

        try {
            setLoading(true);
```

```
            await createUserWithEmailAndPassword(auth,
emailRef.current.value, passwordRef.current.value);
            history('/');
        } catch (err) {
            setError('Failed to Register');
            console.log(err);
            window.alert('Failed to register');
            passwordRef.current.value = '';
            passwordConfirmRef.current.value = '';
        } finally {
            setLoading(false);
        }
    }
```

LOGOUT AND OTHER OPERATIONS ON TASK LOGIC:

```
const [todo, setTodo] = useState("");

  const [todos, setTodos] = useState([]);
  const [filterStatus, setFilterStatus] = useState("all");
  const [priority, setPriority] = useState();
  const navigate = useNavigate();
  const [error, setError] = useState("");
  const { currentUser, logout } = useAuth();

  useEffect(() => {
    auth.onAuthStateChanged((user) => {
      if (user) {
        onValue(ref(db, `/${auth.currentUser.uid}`), (snapshot) => {
          setTodos([]);
          const data = snapshot.val();
          if (data !== null) {
            const todoList = Object.values(data);
            setTodos(todoList);
          }
        });
      } else if (!user) {
        navigate("/");
      }
    });
  }, []);

  async function handleSignOut() {
    setError("")
```

```
  try {
    await logout()
    navigate('/')
  } catch {
    setError("Failed to log out")
  }
}
const writeToDatabase = (event) => {
  event.preventDefault();
  if (todo !== "" && priority !== undefined) {
    const uidd = push(ref(db, `/${auth.currentUser.uid}`)).key;
    set(ref(db, `/${auth.currentUser.uid}/${uidd}`), {
      todo: todo,
      completed: false,
      priority: priority,
      uidd: uidd,
    });

    setTodo("");
    setPriority(1);
  } else {

    console.error("Todo or priority is missing.");
  }
};


const handleDelete = (uidd) => {
  remove(ref(db, `/${auth.currentUser.uid}/${uidd}`));
};
const handleComplete = (uidd) => {
  const todoToUpdate = todos.find((todo) => todo.uidd === uidd);
  update(ref(db, `/${auth.currentUser.uid}/${uidd}`), {
    completed: !todoToUpdate.completed,
  });
};

const handleClearCompleted = () => {
  todos.forEach((todo) => {
    if (todo.completed) {
      remove(ref(db, `/${auth.currentUser.uid}/${todo.uidd}`));
    }
  });
};
```