React (Vite) + Node server on Firebase App Hosting

This document describes how to prepare, run, and deploy a React (Vite) front-end served by a small Node/Express app to **Firebase App Hosting**. It also covers local environment variables, Firebase Auth & Firestore wiring, and App Hosting build settings.

---

0) Prerequisites

- **Node.js 20.x** installed locally (`node -v` → `v20.*`).
- **Git** and a **GitHub** account.
- A **Firebase project** (Console → Add project).
- (Optional for local DX) **VS Code**.

> On Windows PowerShell, run programs from an *elevated* prompt if permission issues arise.

---

1) Project layout (minimal)

Your repository (root) should look like:

```
.
├─ src/
│  ├─ main.jsx
│  └─ App.jsx          ← your React app
├─ public/
│  └─ vite.svg
├─ index.html
├─ package.json
├─ server.js           ← tiny Node/Express to serve dist in production
├─ vite.config.mjs
├─ .gitignore
└─ (optional) Tailwind: postcss.config.cjs, tailwind.config.cjs, src/
index.css
```

If you already have these files, **keep them**. If not, use the examples below.

1.1 `package.json`

```
{
  "name": "imci-app",
  "private": true,
  "type": "module",
```

```
  "engines": { "node": ">=20" },
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.19.2",
    "react": "^19.1.0",
    "react-dom": "^19.1.0"
  },
  "devDependencies": {
    "@vitejs/plugin-react": "^5.0.0",
    "vite": "^7.1.1"
  }
}
```

The `"engines"` field helps App Hosting pick a compatible Node; we'll also set an env variable there at deploy time.

1.2 `server.js` (production server)

```
import express from "express";
import path from "path";
import { fileURLToPath } from "url";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const app = express();
const distDir = path.join(__dirname, "dist");

// gzip/static
app.use(express.static(distDir, { index: false }));

// SPA fallback to index.html
app.get("*", (req, res) => {
  res.sendFile(path.join(distDir, "index.html"));
});

const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server listening on http://localhost:${PORT}`);
});
```

1.3 `vite.config.mjs`

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
  server: { port: 5173, open: true }
});
```

1.4 `.gitignore`

```
node_modules
dist
.DS_Store

# Local env files (keep them out of Git)
.env.local
.env.*.local
```

---

2) Local Firebase configuration

Create a Web App in Firebase Console → **Project settings → General → Your apps → Web App**. Copy the configuration values (apiKey, authDomain, projectId, etc.).

Create `` at the repository root (same folder as `package.json`):

```
VITE_FB_API_KEY=YOUR_API_KEY
VITE_FB_AUTH_DOMAIN=YOUR_AUTH_DOMAIN
VITE_FB_PROJECT_ID=YOUR_PROJECT_ID
VITE_FB_STORAGE_BUCKET=YOUR_STORAGE_BUCKET
VITE_FB_MESSAGING_SENDER_ID=YOUR_SENDER_ID
VITE_FB_APP_ID=YOUR_APP_ID
```

> **Important:** Vite only exposes variables with the `VITE_` prefix.

Restart the dev server whenever you edit env files.

---

3) Wire Firebase in the app

Create `src/firebase.js`:

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";

// Prefer environment (Vite) vars locally; in App Hosting we can fallback to
injected config
const cfg =
  (typeof window !== "undefined" && window.__FIREBASE_WEBAPP_CONFIG__) || {
    apiKey: import.meta.env.VITE_FB_API_KEY,
    authDomain: import.meta.env.VITE_FB_AUTH_DOMAIN,
    projectId: import.meta.env.VITE_FB_PROJECT_ID,
    storageBucket: import.meta.env.VITE_FB_STORAGE_BUCKET,
    messagingSenderId: import.meta.env.VITE_FB_MESSAGING_SENDER_ID,
    appId: import.meta.env.VITE_FB_APP_ID
  };

export const app = initializeApp(cfg);
export const auth = getAuth(app);
export const db = getFirestore(app);
```

Now you can import `auth` / `db` anywhere:

```
import { auth, db } from "./firebase";
// e.g., signInWithEmailAndPassword(auth, email, password)
// e.g., doc(db, "courses", id) / setDoc / getDocs ...
```

---

4) Local development

```
npm install       # first time
npm run dev       # http://localhost:5173
```

If you need Tailwind/PostCSS, add the usual configs; otherwise skip.

---

5) Prepare Firebase (Console)

1. **Enable Authentication**\ Console → **Build** → **Authentication** → **Sign-in method**\ Enable **Email/Password** and/or **Google**.

2. **Enable Firestore**\ Console → **Build** → **Firestore Database** → **Create database** \ Start in **Production**. (You can tighten rules later.)

3. **Authorized domains**\ Authentication → **Settings** → add your **App Hosting domain** (and `localhost` for local dev).

---

6) Put code on GitHub

```
git init
git add .
git commit -m "Initial React+Node app"
git branch -M main
git remote add origin https://github.com/<you>/<repo>.git
git push -u origin main
```

---

7) Firebase App Hosting (deploy from GitHub)

In **Firebase Console → App Hosting**:

1. **Import GitHub repository** (connect your repo and branch).

2. **Deployment settings** (Build & Run) — set four fields:

3. **Install command:** `npm ci`\ (Use `npm install` only if you don't commit a lockfile.)

4. **Build command:** `npm run build`
5. **Start command:** `npm start`

6. **App directory:** `/` (root)

7. **Configure your backend → Runtime**\ Add an **environment variable** to nail Node 20 (lowercase key):

8. **Name:** `bp_node_version`

9. **Value:** `20`

10. **Associate a Firebase web app**\ Select the web app you created earlier.

11. **(Optional) Provide web app config at build time**\ Add an environment variable named `` and paste the JSON Firebase gives you for the web app, e.g.:

```
{"apiKey":"…","authDomain":"…","projectId":"…","storageBucket":"…","messagingSenderId":"…","ap
```

In `src/firebase.js` we read it via `window.__FIREBASE_WEBAPP_CONFIG__` if you choose to inject it (you can expose that in `index.html` or via App Hosting's "runtime env to window" feature if enabled). If not using this mechanism, the `.env.local` is only for local dev; production will use CI-time env injection or direct literals in your code.

1. **Create service** → wait for the GitHub Action to complete.\ When the workflow finishes, your site URL appears in App Hosting.

8) Basic Auth/UI sample (optional)

```
// src/AuthGate.jsx
import React from "react";
import { onAuthStateChanged, signInWithEmailAndPassword, signOut } from
"firebase/auth";
import { auth } from "./firebase";

export default function AuthGate({ children }) {
  const [user, setUser] = React.useState(null);
  const [ready, setReady] = React.useState(false);

  React.useEffect(() => {
    const off = onAuthStateChanged(auth, u => { setUser(u);
setReady(true); });
    return () => off();
  }, []);

  if (!ready) return <div>Loading…</div>;
  if (!user) return <LoginForm />;

  return (
    <div>
      <div className="text-sm">Signed in as {user.email} <button onClick={()
=> signOut(auth)}>Sign out</button></div>
      {children}
    </div>
  );
}

function LoginForm() {
  const [email, setEmail] = React.useState("");
  const [pass, setPass] = React.useState("");
  const [err, setErr] = React.useState("");

  async function submit(e) {
    e.preventDefault();
    setErr("");
    try { await signInWithEmailAndPassword(auth, email, pass); }
    catch (e) { setErr(e.message); }
  }

  return (
    <form onSubmit={submit}>
      <h3>Sign in</h3>
      <input value={email} onChange={e=>setEmail(e.target.value)}
placeholder="email" />
      <input type="password" value={pass}
onChange={e=>setPass(e.target.value)} placeholder="password" />
```

```
      <button type="submit">Sign in</button>
      {err && <div style={{color:'red'}}>{err}</div>}
    </form>
  );
}
```

Then wrap your app in `src/main.jsx`:

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import AuthGate from "./AuthGate.jsx";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <AuthGate>
      <App />
    </AuthGate>
  </React.StrictMode>
);
```

---

9) Firestore example (CRUD)

```
import { db } from "./firebase";
import { collection, addDoc, getDocs, doc, setDoc, deleteDoc } from
"firebase/firestore";

// add
await addDoc(collection(db, "courses"), { name: "IMNCI", state:
"Khartoum" });

// list
const snap = await getDocs(collection(db, "courses"));
const rows = snap.docs.map(d => ({ id: d.id, ...d.data() }));

// update
await setDoc(doc(db, "courses", rows[0].id), { state: "Gezira" }, { merge:
true });

// delete
await deleteDoc(doc(db, "courses", rows[0].id));
```

Organize collections as you prefer (e.g., `courses/{courseId}/participants`, `observations`, etc.).

---

10) Local → Firestore migration (optional)

If you previously stored data in `localStorage`, create `src/migrate.js` with a function that reads those keys and writes documents to Firestore. Temporarily add a button in your app (coordinators only) to call it **once**. After verifying data in Firestore, remove the button and the file.

(If you'd like, I can produce the exact migration code tailored to your keys.)

---

11) Troubleshooting

- **Build fails: "Cannot find module @rollup/rollup-linux-x64-gnu"**\ This is an npm optional-deps quirk. In App Hosting, delete the lockfile and let CI recreate it, or re-run the job with `npm ci` (not `npm install`) and a fresh lock.
- **"vite is not recognized" locally**\ Run `npm install` first; ensure `node` and `npm` are on PATH.
- **Tailwind/PostCSS error**\ Install PostCSS deps and use `postcss.config.cjs` & `tailwind.config.cjs`. Example PostCSS:

  ```
  // postcss.config.cjs
  module.exports = { plugins: { tailwindcss: {}, autoprefixer: {} } };
  ```

- **Auth popup blocked**\ Add your domain in Authentication → Settings → Authorized domains.
- **.env not loaded**\ Ensure file name is `.env.local`, variables start with `VITE_`, and restart `npm run dev`.

---

12) Commands cheat-sheet

```
# Local
npm install
npm run dev
npm run build
npm start            # serve dist with Node (like production)

# Git/GitHub
git add .
git commit -m "Update"
git push

# App Hosting – Build & Run fields
#   Install: npm ci
#   Build:   npm run build
#   Start:   npm start
#   App dir: /
# Env var:  bp_node_version = 20
```

You now have a single, reproducible path to run your React app locally and deploy it behind a Node server on **Firebase App Hosting**, with Auth/Firestore ready to use. If you want me to add the **exact** Firestore collections (schema) and a **migration script** for your existing keys (`imci_courses_v9`, `imci_participants_v9`, `imci_observations_v9`, `imci_cases_v2`), tell me and I'll include the code blocks you can drop in.

Appendix A — Minimal additions to `src/App.jsx` (wire Firestore with very few edits)

> These are **drop-in additions** you can paste into your existing `src/App.jsx`. They keep your current localStorage behavior and **mirror** data to Firestore in the background. You can later turn off localStorage by removing the `persist(...)` calls if desired.

A.1 Imports (top of file)

```
// === Firestore (ADD at the very top of App.jsx)
import { db } from "./firebase";
import {
  collection,
  doc,
  getDocs,
  setDoc,
  deleteDoc,
  serverTimestamp
} from "firebase/firestore";
```

A.2 Feature flag (near your config/constants)

```
// Toggle Firestore sync (true to use Firestore alongside localStorage)
const USE_FIRESTORE = true; // set false to disable

// Firestore collection names
const COLS = {
  courses: "courses",
  participants: "participants",
  observations: "observations",
  cases: "cases"
};
```

A.3 Generic helpers (place below your small helpers or before the root App component)

```
// ---- Firestore helpers (non-blocking best-effort mirror) ----
async function fsSyncCollection(colName, arr) {
  if (!USE_FIRESTORE) return;
  try {
```

```
    const colRef = collection(db, colName);
    const ops = arr.map((it) =>
      setDoc(doc(colRef, String(it.id ?? it.caseId ?? cryptoRand())),
        { ...it, _updatedAt: serverTimestamp() },
        { merge: true }
      )
    );
    await Promise.allSettled(ops);
  } catch (e) {
    console.warn("[FS] sync", colName, e);
  }
}

async function fsDelete(colName, id) {
  if (!USE_FIRESTORE) return;
  try { await deleteDoc(doc(collection(db, colName), String(id))); }
  catch (e) { console.warn("[FS] del", colName, id, e); }
}

function cryptoRand() {
  // tiny id if some legacy rows missed `id`
  return Math.random().toString(36).slice(2, 10);
}
```

A.4 One-time Firestore bootstrap load (prefer Firestore if it already has data)

Paste these two `useEffect` hooks **inside** your root `App` function, right after the existing `useState` declarations for `courses`, `participants` and **before** `return (...)`:

```
// 1) On first load, if Firestore has data for each collection, prefer it
over LS
React.useEffect(() => {
  if (!USE_FIRESTORE) return;
  (async () => {
    try {
      const snap = await getDocs(collection(db, COLS.courses));
      if (!snap.empty) {
        const rows = snap.docs.map(d => d.data());
        setCourses(rows);
      }
    } catch (e) { console.warn("[FS] load courses", e); }
  })();
}, []);

React.useEffect(() => {
  if (!USE_FIRESTORE) return;
  (async () => {
    try {
      const snap = await getDocs(collection(db, COLS.participants));
      if (!snap.empty) {
```

```
        const rows = snap.docs.map(d => d.data());
        setParticipants(rows);
      }
    } catch (e) { console.warn("[FS] load participants", e); }
  })();
}, []);
```

Observations and cases can be heavy; we keep them local until you migrate (see A.6) and then enable loading in the same way if needed.

A.5 Mirror changes to Firestore (non-blocking)

Add these **after** your existing `useEffect(() => persist(...), [...])` hooks so local behavior stays the same:

```
// Mirror to Firestore whenever arrays change
React.useEffect(() => { fsSyncCollection(COLS.courses, courses); },
[courses]);
React.useEffect(() => { fsSyncCollection(COLS.participants,
participants); }, [participants]);

// If you want to mirror observations/cases continuously too:
// const allObs = restore(LS_OBS, []);
// React.useEffect(() => { fsSyncCollection(COLS.observations, allObs); },
[allObs]);
```

If you have specific delete flows (e.g., deleting a case), call `fsDelete(...)` alongside your current state updates, for example inside `deleteCase`:

```
// Inside deleteCase(c)
fsDelete(COLS.observations, `${c.date}|${c.setting}|${c.age}|${c.serial}
`); // or the row ids you use
```

A.6 One-click migration from localStorage → Firestore

Add this helper (anywhere above your component `return`) and a temporary button on the Landing or Reports view. It lifts **your exact keys**:

```
async function migrateLocalToFirestore() {
  if (!USE_FIRESTORE) { alert("Enable USE_FIRESTORE first."); return; }
  const coursesLS = restore(LS_COURSES, []);
  const partsLS   = restore(LS_PARTS,   []);
  const obsLS     = restore(LS_OBS,     []);
  const casesLS   = restore(LS_CASES,   []);

  await fsSyncCollection(COLS.courses, coursesLS);
  await fsSyncCollection(COLS.participants, partsLS);
```

```
  await fsSyncCollection(COLS.observations, obsLS);
  await fsSyncCollection(COLS.cases, casesLS);

  alert("Migration queued. Check Firestore collections.");
}
```

Temporary UI button (e.g., show only to admins/coordinators). Add somewhere convenient (Landing header or Reports):

```
<button
  className="px-3 py-2 rounded-xl border"
  onClick={migrateLocalToFirestore}
>
  Migrate local → Firestore
</button>
```

> After verifying data in Firestore, you may remove the button and optionally delete the local `persist(...)` calls to make Firestore the single source of truth.

A.7 Notes on ids

- Your existing objects already carry `id` (e.g., for courses/participants). Firestore uses the **document id** we set from that `id`, so reads/writes remain stable.
- For observations/cases where a natural id isn't present, you can compose one (e.g., `${participant_id}|${encounter_date}|${case_serial}`) and use that string when calling `doc()`.

Appendix B — Optional: Tighten Firestore Security Rules (outline)

When you move fully to Firestore, add rules in **Firestore → Rules**. A simple starter (require auth and basic shape) might look like:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    function isSignedIn() { return request.auth != null; }

    match /courses/{id}       { allow read, write: if isSignedIn(); }
    match /participants/{id}  { allow read, write: if isSignedIn(); }
    match /observations/{id}  { allow read, write: if isSignedIn(); }
    match /cases/{id}         { allow read, write: if isSignedIn(); }
  }
}
```

Refine per your program's roles later (e.g., facilitators/coordinators).