

# Fullstack Notes App Project

The whole course in one build: a notes app with a React front end, a Node/Express backend, MongoDB storage, register/login with JWT, and notes only you can see.

## Introduction

---

Every lesson so far built one muscle. This capstone project uses them all at once: a real **notes app** — sign up, log in, write notes, edit them, delete them — with React in the browser, a Node/Express API behind it (Node series), and MongoDB holding the data (MongoDB series). It is the shape of nearly every real web product, shrunk to one honest build. (*Roman Urdu: Yeh poori course ka imtihaan hai — front end, back end, database, login, sab aik hi app mein.*)

## Key points

---

- **The architecture is three layers with two contracts:**
  - **React** (browser): screens, forms, state — talks HTTP only.
  - **Express API** (server): routes like `POST /api/notes` — checks the login token, talks to the database.
  - **MongoDB**: users and notes collections — every note carries its owner's id.
- **Auth runs on JWT**: register saves the user with a **hashed** password (never plain text); login verifies and returns a signed **token**; the front end stores it and sends it with every request (`Authorization: Bearer ...`); the server's middleware checks it before any notes route runs.  
(*Roman Urdu: Token tumhari pehchan ki parchi hai — har request ke saath dikhani parti hai.*)
- **"Only your own notes" is a query condition**: `Note.find({ owner: userId })` — the filter is applied on the server from the token's identity, never from anything the browser claims. This line is the app's whole privacy model.
- **CRUD maps to HTTP verbs you know**: Create = POST, Read = GET (list + one), Update = PUT, Delete = DELETE — the same table the Node and database series taught, now with a real UI on top.
- **The front end assembles the course**: Router (5.18) splits Login/Register/Notes pages, a protected route guards the notes page, React Query or the fetch pattern (5.19/5.22) moves data, forms are controlled components (5.6), and the token lives in state/context (5.7).
- **Environment split**: front end on Netlify/Vercel, the API on a Node host, MongoDB Atlas free tier — all free, all real; the API URL reaches the front end through the env variables of 5.24.

## Code examples

---

### Example 1 — The backend heart: a protected notes route

```
// Express — GET /api/notes (auth middleware ran first)
app.get("/api/notes", auth, async (req, res) => {
  const notes = await Note.find({ owner: req.user.id }).sort({ updatedAt: -1 });
  res.json(notes);
});

app.post("/api/notes", auth, async (req, res) => {
  const note = await Note.create({
    title: req.body.title,
    body: req.body.body,
    owner: req.user.id, // from the verified token — never from the request body
  });
  res.status(201).json(note);
});
```

What it does: The `auth` middleware verifies the JWT and attaches `req.user`. Reading returns only the caller's notes; creating stamps the new note with the caller's id from the token — if a request tried to smuggle another `owner` in its body, it is ignored. The video builds the matching update/delete routes and shows two accounts unable to see each other's notes.

**What you will see:** log in as two different users in two browsers — each sees an empty list, adds notes, and never glimpses the other's, though both live in the same collection.

### Example 2 — The frontend: a protected route for the notes page

```
import { Navigate } from "react-router-dom";
import { useAuth } from "../AuthContext";

function ProtectedRoute({ children }) {
  const { user } = useAuth();
  if (!user) return <Navigate to="/login" replace />;
  return children;
}

// In the route table:
// <Route path="/notes" element={
//   <ProtectedRoute><NotesPage /></ProtectedRoute>
// } />
```

What it does: The login state lives in an AuthContext (5.7's pattern) fed by the stored token. The protected route is the conditional rendering of 5.15 at app scale: no user, no page — a redirect to Login instead.

Logging out clears the token and the same guard quietly closes the door behind you.

**What you will see:** paste the `/notes` URL into a fresh browser without logging in — you land on Login. Log in, and the same URL opens your notes.

### Example 3 — The notes page: list, add, delete against the API

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

const API = import.meta.env.VITE_API_URL;

function NotesPage() {
  const [notes, setNotes] = useState([]);
  const [title, setTitle] = useState("");
  const token = localStorage.getItem("token");
  const headers = {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`,
  };

  useEffect(() => {
    fetch(`${API}/api/notes`, { headers })
      .then((r) => r.json())
      .then(setNotes);
  }, []);

  async function addNote(e) {
    e.preventDefault();
    const res = await fetch(`${API}/api/notes`, {
      method: "POST",
      headers,
      body: JSON.stringify({ title, body: "" }),
    });
    const saved = await res.json();
    setNotes((list) => [saved, ...list]);
    setTitle("");
  }

  async function removeNote(id) {
    await fetch(`${API}/api/notes/${id}`, { method: "DELETE", headers });
    setNotes((list) => list.filter((n) => n._id !== id));
  }

  return (
    <div>
      <form onSubmit={addNote}>
        <input value={title} onChange={(e) => setTitle(e.target.value)} placeholder="New note title" />
        <button>Add note</button>
      </form>
    </div>
  );
}
```

```

    <ul>
      {notes.map((n) => (
        <li key={n._id}>
          {n.title}
          <button onClick={() => removeNote(n._id)}>Delete</button>
        </li>
      ))}
    </ul>
  </div>
);
}

```

What it does: Every piece of the course in one screen: the token header built once and reused, the mount fetch (5.19), the add-without-refetch pattern, delete-by-id against both the API and the local state, MongoDB's `_id` serving as the list key (5.16).

**What you will see:** add a note — it appears; refresh — it is still there (it lives in MongoDB now, not in your browser); delete it — gone from screen and database together.

## Quick recap

---

- The capstone is three layers — **React** ↔ **Express API** ↔ **MongoDB** — joined by HTTP and guarded by **JWT auth**: hashed passwords, a token per login, middleware on every notes route.
- Privacy is one server-side habit: **every query filters by the token's user id**.
- Nothing in this build is new — Router, context, fetch patterns, forms and deployment are the earlier lessons, assembled.

## Common mistakes

---

1. **Trusting the browser for identity** — accepting `owner` or `userId` from the request body lets anyone read or write anyone's notes by editing a request. The fix: identity comes from the verified token on the server, only.
2. **Storing plain-text passwords** — one database leak exposes every user's password everywhere they reused it. The fix: hash with bcrypt at register, compare hashes at login; the plain password is never saved.
3. **Sending the token but never checking expiry/errors on the front end** — a stale token leaves the app showing empty lists forever with no explanation. The fix: a 401 from the API means "log out and show Login" — handle it in one shared place.

## Practice

---

1. Complete the journey: add the Edit flow — clicking a note loads it into the form, saving sends `PUT /api/notes/:id`, and the list updates from the server's answer. Add the matching Express route with the

same owner filter on the update (find the note by id **and** owner together).

2. Deploy the whole stack free: MongoDB Atlas for the database, the API on a free Node host, the front end on Netlify with `VITE_API_URL` set. Then do the 5.24 stranger test as a brand-new user: register, add two notes, log out, log back in — on your phone.

---

**Watch the Code Scope channel:** <https://www.youtube.com/@CodeScopePK> — this lesson's own video joins the channel when it is recorded.

**Read online:** <https://toolscope.netlify.app/notes/react-fullstack-notes-project> · More free tools and notes at <https://toolscope.netlify.app>