**Backend — FastAPI (complete)**

backend/app/models.py

from typing import Optional

from sqlmodel import SQLModel, Field

import uuid

from datetime import datetime

class Note(SQLModel, table=True):

id: Optional[int] = Field(default=None, primary\_key=True)

title: str

content: str

created\_at: datetime = Field(default\_factory=datetime.utcnow)

updated\_at: datetime = Field(default\_factory=datetime.utcnow)

public\_id: Optional[str] = None # uuid string for shareable link

backend/app/database.py

from sqlmodel import SQLModel, create\_engine, Session

import os

DATABASE\_URL = os.getenv("DATABASE\_URL", "sqlite:///./notes.db")

# For Render/GCP/Heroku, you can set DATABASE\_URL to a Postgres URL if desired.

connect\_args = {}

if DATABASE\_URL.startswith("sqlite"):

connect\_args = {"check\_same\_thread": False}

engine = create\_engine(DATABASE\_URL, echo=False, connect\_args=connect\_args)

def init\_db():

SQLModel.metadata.create\_all(engine)

def get\_session():

return Session(engine)

backend/app/crud.py

from sqlmodel import select

from .models import Note

from .database import get\_session

from typing import List, Optional

import uuid

from datetime import datetime

def create\_note(title: str, content: str) -> Note:

with get\_session() as s:

n = Note(title=title, content=content, created\_at=datetime.utcnow(), updated\_at=datetime.utcnow())

s.add(n)

s.commit()

s.refresh(n)

return n

def get\_notes() -> List[Note]:

with get\_session() as s:

return s.exec(select(Note).order\_by(Note.updated\_at.desc())).all()

def get\_note(note\_id: int) -> Optional[Note]:

with get\_session() as s:

return s.get(Note, note\_id)

def update\_note(note\_id: int, title: str, content: str) -> Optional[Note]:

with get\_session() as s:

n = s.get(Note, note\_id)

if not n:

return None

n.title = title

n.content = content

n.updated\_at = datetime.utcnow()

s.add(n)

s.commit()

s.refresh(n)

return n

def delete\_note(note\_id: int) -> bool:

with get\_session() as s:

n = s.get(Note, note\_id)

if not n:

return False

s.delete(n)

s.commit()

return True

def share\_note(note\_id: int) -> Optional[str]:

with get\_session() as s:

n = s.get(Note, note\_id)

if not n:

return None

if not n.public\_id:

n.public\_id = str(uuid.uuid4())

s.add(n)

s.commit()

s.refresh(n)

return n.public\_id

def get\_note\_by\_public\_id(public\_id: str) -> Optional[Note]:

with get\_session() as s:

statement = select(Note).where(Note.public\_id == public\_id)

return s.exec(statement).first()

backend/app/main.py

from fastapi import FastAPI, HTTPException

from fastapi.middleware.cors import CORSMiddleware

from pydantic import BaseModel

from typing import List

from . import crud, database, models

app = FastAPI(title="Notes API")

database.init\_db()

# adjust origins to your frontend URL in deployment

origins = [

"http://localhost:5173",

"http://localhost:3000",

# add your Vercel domain(s) here or set to "\*" for quick tests

]

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # for production set to your actual frontend URL(s)

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

class NoteIn(BaseModel):

title: str

content: str

class NoteOut(BaseModel):

id: int

title: str

content: str

created\_at: str

updated\_at: str

public\_id: str | None

@app.get("/health")

def health():

return {"ok": True}

@app.get("/notes", response\_model=List[NoteOut])

def read\_notes():

return crud.get\_notes()

@app.post("/notes", response\_model=NoteOut)

def create\_note(n: NoteIn):

note = crud.create\_note(n.title, n.content)

return note

@app.get("/notes/{note\_id}", response\_model=NoteOut)

def read\_note(note\_id: int):

note = crud.get\_note(note\_id)

if not note:

raise HTTPException(status\_code=404, detail="Note not found")

return note

@app.put("/notes/{note\_id}", response\_model=NoteOut)

def put\_note(note\_id: int, n: NoteIn):

note = crud.update\_note(note\_id, n.title, n.content)

if not note:

raise HTTPException(status\_code=404, detail="Note not found")

return note

@app.delete("/notes/{note\_id}")

def remove\_note(note\_id: int):

ok = crud.delete\_note(note\_id)

if not ok:

raise HTTPException(status\_code=404, detail="Note not found")

return {"ok": True}

@app.post("/notes/{note\_id}/share")

def share(note\_id: int):

public\_id = crud.share\_note(note\_id)

if not public\_id:

raise HTTPException(status\_code=404, detail="Note not found")

return {"public\_id": public\_id}

@app.get("/public/{public\_id}")

def public\_view(public\_id: str):

note = crud.get\_note\_by\_public\_id(public\_id)

if not note:

raise HTTPException(status\_code=404, detail="Public note not found")

return {

"id": note.id,

"title": note.title,

"content": note.content,

"created\_at": note.created\_at,

"updated\_at": note.updated\_at

}

backend/requirements.txt

fastapi

uvicorn[standard]

sqlmodel

pydantic

backend/Dockerfile

FROM python:3.11-slim

WORKDIR /app

COPY ./app /app/app

COPY requirements.txt /app/requirements.txt

RUN pip install --no-cache-dir -r /app/requirements.txt

ENV PORT=8000

EXPOSE $PORT

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

backend/start.sh (optional)

#!/bin/sh

uvicorn app.main:app --host 0.0.0.0 --port ${PORT:-8000}

Notes:

* The backend uses SQLite ./notes.db by default. On Render, the ephemeral filesystem may be reset between deploys — for production use, attach a managed Postgres DB and set DATABASE\_URL accordingly. I’ll include guidance below.

**Frontend — React (Vite) — key files**

frontend/package.json (relevant bits)

{

"name": "notes-frontend",

"version": "1.0.0",

"scripts": {

"dev": "vite",

"build": "vite build",

"preview": "vite preview"

},

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0"

},

"devDependencies": {

"vite": "^5.0.0"

}

}

frontend/src/api.js

const API = import.meta.env.VITE\_API\_URL || "http://localhost:8000";

export async function fetchNotes(){

const res = await fetch(`${API}/notes`);

return res.json();

}

export async function createNote(data){

const res = await fetch(`${API}/notes`, {

method: 'POST',

headers:{'Content-Type':'application/json'},

body: JSON.stringify(data)

});

return res.json();

}

export async function updateNote(id, data){

const res = await fetch(`${API}/notes/${id}`, {

method: 'PUT',

headers:{'Content-Type':'application/json'},

body: JSON.stringify(data)

});

return res.json();

}

export async function deleteNote(id){

const res = await fetch(`${API}/notes/${id}`, { method: 'DELETE' });

return res.json();

}

export async function shareNote(id){

const res = await fetch(`${API}/notes/${id}/share`, { method: 'POST' });

return res.json();

}

export async function fetchPublic(public\_id){

const res = await fetch(`${API}/public/${public\_id}`);

return res.json();

}

frontend/src/App.jsx

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

import { fetchNotes, createNote, updateNote, deleteNote, shareNote } from "./api";

import NoteList from "./components/NoteList";

import NoteEditor from "./components/NoteEditor";

export default function App(){

const [notes, setNotes] = useState([]);

const [editing, setEditing] = useState(null);

async function load(){

const data = await fetchNotes();

setNotes(data);

}

useEffect(()=>{ load() }, []);

async function handleCreate(title, content){

await createNote({title, content});

await load();

}

async function handleUpdate(id, title, content){

await updateNote(id, {title, content});

setEditing(null);

await load();

}

async function handleDelete(id){

await deleteNote(id);

await load();

}

async function handleShare(id){

const res = await shareNote(id);

// Build a public URL pointing at frontend public view

const apiBase = import.meta.env.VITE\_API\_URL || "http://localhost:8000";

const frontendBase = window.location.origin;

const publicURL = `${frontendBase}/public/${res.public\_id}`;

// copy to clipboard

navigator.clipboard.writeText(publicURL);

alert("Public link copied to clipboard:\n" + publicURL);

}

return (

<div style={{padding:20, maxWidth:900, margin:"0 auto"}}>

<h1>Notes App</h1>

<NoteEditor onSave={handleCreate} />

<hr />

<NoteList notes={notes} onEdit={setEditing} onDelete={handleDelete} onShare={handleShare} />

{editing && <NoteEditor note={editing} onSave={(t,c)=>handleUpdate(editing.id,t,c)} onCancel={()=>setEditing(null)} />}

</div>

);

}

frontend/src/components/NoteList.jsx

import React from "react";

export default function NoteList({notes, onEdit, onDelete, onShare}){

return (

<div>

{notes.length === 0 && <p>No notes yet</p>}

{notes.map(n => (

<div key={n.id} style={{border:"1px solid #ddd", padding:12, marginBottom:10, borderRadius:6}}>

<h3>{n.title}</h3>

<p style={{whiteSpace:"pre-wrap"}}>{n.content}</p>

<div style={{display:"flex", gap:8}}>

<button onClick={()=>onEdit(n)}>Edit</button>

<button onClick={()=>onDelete(n.id)}>Delete</button>

<button onClick={()=>onShare(n.id)}>Share</button>

</div>

</div>

))}

</div>

);

}

frontend/src/components/NoteEditor.jsx

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

export default function NoteEditor({note, onSave, onCancel}){

const [title, setTitle] = useState(note?.title || "");

const [content, setContent] = useState(note?.content || "");

useEffect(()=>{

setTitle(note?.title || "");

setContent(note?.content || "");

}, [note]);

return (

<div style={{marginBottom:20}}>

<input value={title} onChange={e=>setTitle(e.target.value)} placeholder="Title" style={{width:"100%", padding:8, marginBottom:8}} />

<textarea value={content} onChange={e=>setContent(e.target.value)} placeholder="Write your note..." rows={6} style={{width:"100%", padding:8}} />

<div style={{marginTop:8}}>

<button onClick={()=>onSave(title, content)}>{note ? "Update" : "Create"}</button>

{onCancel && <button onClick={onCancel} style={{marginLeft:8}}>Cancel</button>}

</div>

</div>

);

}

frontend/src/components/PublicView.jsx (optional — load public note into frontend route /public/:public\_id)

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

import { fetchPublic } from "../api";

import { useParams } from "react-router-dom";

export default function PublicView(){

const { publicId } = useParams();

const [note, setNote] = useState(null);

useEffect(()=>{

async function load(){

const res = await fetchPublic(publicId);

setNote(res);

}

load();

}, [publicId]);

if(!note) return <div>Loading...</div>;

return (

<div style={{maxWidth:800, margin:"auto", padding:20}}>

<h1>{note.title}</h1>

<div style={{whiteSpace:"pre-wrap"}}>{note.content}</div>

</div>

);

}

frontend/vercel.json (optional)

{

"builds": [{ "src": "package.json", "use": "@vercel/static-build", "config": { "distDir": "dist" } }],

"routes": [

{ "src": "/(.\*)", "dest": "/" }

]

}

Important: in Vite, set VITE\_API\_URL in Vercel environment variables to your deployed backend URL, e.g. https://your-backend.onrender.com.

**How sharing works (flow)**

1. User clicks **Share** on a note.
2. Frontend calls POST /notes/{id}/share — backend generates (or returns existing) public\_id (UUID).
3. Frontend constructs a **public URL**: https://<frontend-domain>/public/<public\_id> (or you can share the backend public GET: https://<backend>/public/<public\_id>; both work).
4. Anyone with that URL can view the note (via frontend public route or directly fetch backend /public/{public\_id}).

**Deployment instructions — Backend (Render example)**

1. Push the backend/ folder to GitHub (or place entire repo and let Render pick the right folder).
2. On Render:
   * Create a new **Web Service**.
   * Connect your GitHub repo.
   * Set the build command: (if using Dockerfile, Render will auto-detect). Or use:
     + pip install -r requirements.txt and start command: uvicorn app.main:app --host 0.0.0.0 --port $PORT
   * Set the environment variable PORT if required (Render sets it automatically).
   * For data persistence: either:
     + Use SQLite (quick demo) — the file will be created but note: Render's filesystem is ephemeral across deploys. For stable persistence, create a managed Postgres database on Render and set DATABASE\_URL=postgres://....
3. If using Dockerfile (recommended for deterministic build), Render will use it automatically.
4. Deploy and note the public URL (e.g., https://notes-backend.onrender.com). This is your backend API URL.

Notes about DATABASE:

* For production persistence, create a managed Postgres database, set DATABASE\_URL in Render, and adjust connect\_args (SQLModel works with Postgres, no special connect\_args needed).

**Deployment instructions — Frontend (Vercel)**

1. Push frontend/ to GitHub (or repo root).
2. On Vercel:
   * Create a new project from GitHub.
   * For build & output settings (Vite should be auto-detected):
     + Build Command: npm run build
     + Output Directory: dist
   * Important: add Environment Variable VITE\_API\_URL and set it to your backend API URL (e.g., https://notes-backend.onrender.com).
   * Deploy. Vercel will supply a frontend URL like https://notes-frontend.vercel.app.
3. Open the frontend URL; it will call the backend API URL you set.

**Quick local testing (before deploying)**

* Backend:
* cd backend
* python -m venv .venv
* . .venv/bin/activate
* pip install -r requirements.txt
* uvicorn app.main:app --reload
* Frontend:
* cd frontend
* npm install
* VITE\_API\_URL=http://localhost:8000 npm run dev
* Open http://localhost:5173 (Vite default) and test CRUD/share.

**Security & production notes**

* CORS: currently allow\_origins=["\*"] in the example for convenience. For production, restrict to exact frontend URL(s).
* Database: use Postgres for real persistence. Set DATABASE\_URL to Postgres connection string.
* Access control: public links are intentionally public; do not include sensitive data in notes you share.
* Rate limiting / auth: not included in this simple demo.

**What I can do next**

* If you paste your GitHub repo link here I can:
  + Review the repo for deployment issues.
  + Give exact environment variables and an adjusted vercel.json / render.yaml.
* If you want, I can produce a full single-repo ZIP (complete code files placed correctly). I can paste more full file contents if you want them copy/paste-ready.

**Summary / Checklist to finish**

1. Copy the code above into backend/ and frontend/ folders in a GitHub repo.
2. Deploy backend to Render (or Railway, Fly.io, Heroku); record backend URL (e.g., https://<your-backend>.onrender.com).
3. Set VITE\_API\_URL in Vercel to that backend URL and deploy frontend to Vercel.
4. Provide the two URLs (frontend app URL, backend API URL) — those are the final submission items.