**0) Un fichier TDMS d’exemple**

Télécharge un TDMS public pour tester :

# Exemple universitaire (contient un TDMS + description)

curl -L -o sample.tdms "https://www-personal.acfr.usyd.edu.au/zubizarreta/EDatabase.php?file=TDMS"

Si ce lien change, cherche “TDMS sample file” : l’outil **npTDMS** lit ce format, et NI documente la structure (Fichier → Groupes → Canaux). [www-personal.acfr.usyd.edu.au](https://www-personal.acfr.usyd.edu.au/zubizarreta/EDatabase.php?utm_source=chatgpt.com)[nptdms.readthedocs.io](https://nptdms.readthedocs.io/en/stable/reading.html?utm_source=chatgpt.com)[NI](https://www.ni.com/en/support/documentation/supplemental/07/tdms-file-format-internal-structure.html?srsltid=AfmBOoqJ6wwI116ZdAtJCbZHI-WZNLn2aHin-JVwvQouVoBxPAFfMMLa&utm_source=chatgpt.com)

**1) Backend d’ingestion & API (FastAPI + npTDMS)**

**Pourquoi ces choix ?**

* **Python/FastAPI + npTDMS** : écosystème TDMS mûr, lecture rapide. [nptdms.readthedocs.io](https://nptdms.readthedocs.io/?utm_source=chatgpt.com)
* **Stockage** :
  + **Parquet** (fichiers) pour les données brutes converties (colonnaire, compressé). [DuckDB](https://duckdb.org/docs/stable/data/parquet/overview.html?utm_source=chatgpt.com)
  + **SQLite (SQLModel)** pour les **métadonnées** (simple et suffisant pour démarrer).
  + Tu pourras passer à DuckDB/ClickHouse/TimescaleDB plus tard sans changer l’API. [DuckDB](https://duckdb.org/docs/stable/guides/file_formats/query_parquet.html?utm_source=chatgpt.com)[tigerdata.com](https://www.tigerdata.com/blog/simplified-time-series-analytics-using-the-time_bucket-function?utm_source=chatgpt.com)

**Arborescence**

tdms-backend/

app/

main.py

models.py

io\_tdms.py

data/ # Parquet sort ici

db.sqlite

requirements.txt

**requirements.txt**

fastapi==0.115.0

uvicorn[standard]==0.30.6

nptdms==1.10.0

pandas==2.2.2

pyarrow==17.0.0

sqlmodel==0.0.22

python-multipart==0.0.9

UploadFile/multipart est natif dans FastAPI, doc ici. [FastAPI+1](https://fastapi.tiangolo.com/tutorial/request-files/?utm_source=chatgpt.com)

**app/models.py**

from sqlmodel import SQLModel, Field

from typing import Optional, List

from datetime import datetime

class Dataset(SQLModel, table=True):

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

filename: str

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

class Channel(SQLModel, table=True):

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

dataset\_id: int = Field(foreign\_key="dataset.id")

group\_name: str

channel\_name: str

n\_rows: int

parquet\_path: str

has\_time: bool

unit: Optional[str] = None

**app/io\_tdms.py**

from nptdms import TdmsFile

import pandas as pd

import numpy as np

import pyarrow as pa

import pyarrow.parquet as pq

from pathlib import Path

def tdms\_to\_parquet(tdms\_path: str, out\_dir: str):

tdms = TdmsFile.read(tdms\_path)

out = Path(out\_dir)

out.mkdir(parents=True, exist\_ok=True)

meta = []

for group in tdms.groups():

for ch in group.channels():

# Valeurs

values = ch[:]

df = pd.DataFrame({"value": values})

# Temps si dispo

has\_time = False

try:

t = ch.time\_track()

if t is not None:

df.insert(0, "time", pd.to\_datetime(t))

has\_time = True

except Exception:

# fallback : index échantillon -> entier

df.insert(0, "time", np.arange(len(df)))

# Unités si dispo

unit = ch.properties.get("NI\_UnitDescription") or ch.properties.get("unit\_string")

# Nom de fichier parquet

safe = f"{group.name}\_\_{ch.name}".replace("/", "\_")

pq\_path = out / f"{safe}.parquet"

# Ecriture Parquet (compression ZSTD)

table = pa.Table.from\_pandas(df, preserve\_index=False)

pq.write\_table(table, pq\_path, compression="zstd")

meta.append({

"group": group.name,

"channel": ch.name,

"rows": len(df),

"parquet": str(pq\_path),

"has\_time": has\_time,

"unit": unit,

})

return meta

**app/main.py**

from fastapi import FastAPI, UploadFile, File, HTTPException, Query

from fastapi.middleware.cors import CORSMiddleware

from sqlmodel import SQLModel, create\_engine, Session, select

from datetime import datetime

from pathlib import Path

import pandas as pd

import numpy as np

import pyarrow.parquet as pq

from .models import Dataset, Channel

from .io\_tdms import tdms\_to\_parquet

DB\_URL = "sqlite:///db.sqlite"

engine = create\_engine(DB\_URL, connect\_args={"check\_same\_thread": False})

SQLModel.metadata.create\_all(engine)

app = FastAPI(title="TDMS → Parquet API")

# CORS (autorise ton Next.js local)

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], allow\_credentials=True,

allow\_methods=["\*"], allow\_headers=["\*"],

)

DATA\_DIR = Path("data")

DATA\_DIR.mkdir(exist\_ok=True)

@app.post("/ingest")

async def ingest(file: UploadFile = File(...)):

# Sauvegarde temporaire

tmp\_path = Path("tmp") / f"{datetime.utcnow().timestamp()}\_{file.filename}"

tmp\_path.parent.mkdir(exist\_ok=True)

content = await file.read()

tmp\_path.write\_bytes(content)

# Convertit en Parquet + métadonnées

out\_dir = DATA\_DIR / tmp\_path.stem

meta = tdms\_to\_parquet(str(tmp\_path), str(out\_dir))

tmp\_path.unlink()

# Enregistre en DB

with Session(engine) as s:

ds = Dataset(filename=file.filename)

s.add(ds); s.commit(); s.refresh(ds)

for m in meta:

ch = Channel(

dataset\_id=ds.id,

group\_name=m["group"],

channel\_name=m["channel"],

n\_rows=m["rows"],

parquet\_path=m["parquet"],

has\_time=m["has\_time"],

unit=m["unit"],

)

s.add(ch)

s.commit()

return {"dataset\_id": ds.id, "channels": meta}

@app.get("/datasets")

def list\_datasets():

with Session(engine) as s:

rows = s.exec(select(Dataset)).all()

return rows

@app.get("/datasets/{dataset\_id}/channels")

def list\_channels(dataset\_id: int):

with Session(engine) as s:

rows = s.exec(select(Channel).where(Channel.dataset\_id == dataset\_id)).all()

return rows

@app.get("/window")

def get\_window(

channel\_id: int = Query(...),

start: str | None = Query(None, description="ISO datetimes si has\_time, sinon entier minimal"),

end: str | None = Query(None, description="ISO datetimes si has\_time, sinon entier maximal"),

points: int = Query(2000, ge=10, le=20000, description="downsampling cible"),

agg: str = Query("mean", description="mean|max|min")

):

# Récupère métadonnées du canal

with Session(engine) as s:

ch = s.get(Channel, channel\_id)

if not ch:

raise HTTPException(404, "Channel not found")

# Lit Parquet

df = pq.read\_table(ch.parquet\_path).to\_pandas()

# Filtrage fenêtré

if ch.has\_time:

df["time"] = pd.to\_datetime(df["time"])

if start: df = df[df["time"] >= pd.to\_datetime(start)]

if end: df = df[df["time"] <= pd.to\_datetime(end)]

# Downsampling par “bin” temporel (points cibles)

if len(df) > points:

# coupe en 'points' intervalles réguliers

bins = np.linspace(0, len(df)-1, points+1, dtype=int)

take = []

for i in range(len(bins)-1):

seg = df.iloc[bins[i]:bins[i+1]]

if len(seg)==0: continue

if agg == "max": row = seg.loc[seg["value"].idxmax()]

elif agg == "min": row = seg.loc[seg["value"].idxmin()]

else: row = seg.iloc[[0]].assign(value=seg["value"].mean()).iloc[0]

take.append(row)

df = pd.DataFrame(take)

return {

"x": df["time"].astype("datetime64[ms]").dt.strftime("%Y-%m-%dT%H:%M:%S.%fZ").tolist(),

"y": df["value"].astype(float).tolist(),

"unit": ch.unit,

"has\_time": True

}

else:

# Index numérique

if start: df = df[df["time"] >= int(start)]

if end: df = df[df["time"] <= int(end)]

if len(df) > points:

bins = np.linspace(0, len(df)-1, points, dtype=int)

df = df.iloc[bins]

return {

"x": df["time"].astype(int).tolist(),

"y": df["value"].astype(float).tolist(),

"unit": ch.unit,

"has\_time": False

}

**Lancer le serveur**

cd tdms-backend

python -m venv .venv && source .venv/bin/activate

pip install -r requirements.txt

uvicorn app.main:app --reload --port 8000

**Tester l’ingestion**

curl -F "file=@sample.tdms" http://localhost:8000/ingest

curl http://localhost:8000/datasets

curl "http://localhost:8000/datasets/1/channels"

# Récupère une fenêtre downsamplée pour channel\_id=1

curl "http://localhost:8000/window?channel\_id=1&points=1500"

Réfs utiles : **npTDMS** (lecture TDMS), **TDMS** structure par NI, **Parquet/DuckDB** pour la performance. [nptdms.readthedocs.io](https://nptdms.readthedocs.io/en/stable/reading.html?utm_source=chatgpt.com)[NI+1](https://www.ni.com/en/support/documentation/supplemental/07/tdms-file-format-internal-structure.html?srsltid=AfmBOoqJ6wwI116ZdAtJCbZHI-WZNLn2aHin-JVwvQouVoBxPAFfMMLa&utm_source=chatgpt.com)[DuckDB+1](https://duckdb.org/docs/stable/data/parquet/overview.html?utm_source=chatgpt.com)

**2) Frontend Next.js 15 + Plotly.js**

**Installer**

npx create-next-app tdms-frontend --ts --eslint

cd tdms-frontend

npm i react-plotly.js plotly.js

Pour React/Next, react-plotly.js est le wrapper officiel. Sous App Router, on le charge **sans SSR**. [plotly.com](https://plotly.com/javascript/react/?utm_source=chatgpt.com)[GitHub](https://github.com/plotly/react-plotly.js?utm_source=chatgpt.com)[DEV Community](https://dev.to/composite/how-to-integrate-plotlyjs-on-nextjs-14-with-app-router-1loj?utm_source=chatgpt.com)

**app/components/PlotClient.tsx**

"use client";

import dynamic from "next/dynamic";

import { useMemo } from "react";

const Plot = dynamic(() => import("react-plotly.js"), { ssr: false });

export default function PlotClient({ x, y, title }: { x: (string|number)[], y: number[], title: string }) {

const data = useMemo(() => ([

{ x, y, type: "scatter", mode: "lines", name: "signal" }

]), [x, y]);

return (

<Plot

data={data}

layout={{ title, xaxis: { automargin: true }, yaxis: { automargin: true } }}

useResizeHandler

style={{ width: "100%", height: "60vh" }}

config={{ displaylogo: false, responsive: true }}

/>

);

}

**app/page.tsx**

import PlotClient from "./components/PlotClient";

async function fetchJSON(path: string) {

const res = await fetch(process.env.NEXT\_PUBLIC\_API\_BASE + path, { cache: "no-store" });

if (!res.ok) throw new Error("API error");

return res.json();

}

export default async function Page() {

// 1) prends le premier dataset et le premier canal

const datasets = await fetchJSON("/datasets");

if (!datasets?.length) {

return <main className="p-6">Ingest d’abord un fichier TDMS…</main>;

}

const dsId = datasets[0].id;

const channels = await fetchJSON(`/datasets/${dsId}/channels`);

const ch = channels[0];

// 2) récupère une fenêtre downsamplée

const win = await fetchJSON(`/window?channel\_id=${ch.id}&points=2000`);

return (

<main className="p-6">

<h1 className="text-2xl font-semibold mb-4">Dataset {dsId} — {ch.group\_name}/{ch.channel\_name}</h1>

<PlotClient x={win.x} y={win.y} title="Signal downsamplé" />

</main>

);

}

**.env.local**

NEXT\_PUBLIC\_API\_BASE=http://localhost:8000

**Lancer**

npm run dev

# Va sur http://localhost:3000

**3) API design récap (ce que tu viens d’implémenter)**

* POST /ingest : upload TDMS → Parquet + métadonnées (DB).
* GET /datasets : liste des jeux ingérés.
* GET /datasets/{id}/channels : canaux d’un dataset.
* GET /window?channel\_id=...&start=...&end=...&points=...&agg=...  
  → renvoie x[], y[] (downsamplés pour l’affichage).

Pour de très gros volumes ou des besoins analytiques, ajoute un endpoint GET /stats ou passe à Timescale/ClickHouse (agrégations temporelles comme time\_bucket). [tigerdata.com](https://www.tigerdata.com/blog/simplified-time-series-analytics-using-the-time_bucket-function?utm_source=chatgpt.com)

**4) Ressources complémentaires / tutos**

* **Lire TDMS en Python (npTDMS docs)** — exemples de TdmsFile.read() et time\_track(). [nptdms.readthedocs.io+1](https://nptdms.readthedocs.io/en/stable/reading.html?utm_source=chatgpt.com)
* **TDMS format & bonnes pratiques (NI)** — structure hiérarchique et considérations de perf. [NI+2NI+2](https://www.ni.com/en/support/documentation/supplemental/07/tdms-file-format-internal-structure.html?srsltid=AfmBOoqJ6wwI116ZdAtJCbZHI-WZNLn2aHin-JVwvQouVoBxPAFfMMLa&utm_source=chatgpt.com)
* **FastAPI — upload de fichiers** — UploadFile, multipart. [FastAPI+1](https://fastapi.tiangolo.com/tutorial/request-files/?utm_source=chatgpt.com)[Stack Overflow](https://stackoverflow.com/questions/63048825/how-to-upload-file-using-fastapi?utm_source=chatgpt.com)
* **Plotly + React/Next** — wrapper officiel & intégration App Router. [plotly.com](https://plotly.com/javascript/react/?utm_source=chatgpt.com)[GitHub](https://github.com/plotly/react-plotly.js?utm_source=chatgpt.com)[DEV Community](https://dev.to/composite/how-to-integrate-plotlyjs-on-nextjs-14-with-app-router-1loj?utm_source=chatgpt.com)
* **Time series avec Plotly.js** — exemples de lignes/axes temps. [plotly.com](https://plotly.com/javascript/time-series/?utm_source=chatgpt.com)
* **Parquet/DuckDB** — lire/filtrer efficacement (option pro). [DuckDB+2DuckDB+2](https://duckdb.org/docs/stable/data/parquet/overview.html?utm_source=chatgpt.com)

**Variantes / extensions (quand tu seras prêt)**

* **Asynchrone** (RQ/Celery) pour ingérer des fichiers multi-GB, avec GET /jobs/{id} de suivi.
* **Arrow streaming** pour renvoyer des fenêtres ultra-rapides (au lieu de JSON).
* **LTTB** (Largest Triangle Three Buckets) pour un downsampling visuellement fidèle.
* **Sécurité** : auth (Keycloak), quotas d’upload, hash de fichier pour éviter la réingestion.

Si tu veux, je peux te livrer un **repo minimal** (backend + frontend) prêt à cloner avec un docker-compose — dis-moi juste si tu préfères **SQLite** only (dev) ou **Postgres**.