A continuación te dejo **todo lo necesario** para portar la sección *Noticias* del backend Node/Express a tu backend **FastAPI + SQLAlchemy + MySQL**, alineado con tu estructura actual.

---

# 1) Modelo SQLAlchemy (MySQL)

**Archivo:** `app/db/models.py`

> Agregá **esta clase** al final del archivo (no borres nada de lo que ya tenés). Usa los tipos y estilo que ya venís usando (LONGTEXT para contenido, timestamps con `func.now()`).

```
# --- MODELO: Noticias
------------------------------------------------------------
from sqlalchemy.dialects.mysql import LONGTEXT
from sqlalchemy import String, Boolean, DateTime
from sqlalchemy.sql import func
from typing import Optional

class Noticia(Base):
    __tablename__ = "noticias"

    id: Mapped[int] = mapped_column(INTEGER(11), primary_key=True,
autoincrement=True)


# Campos principales (Node: titulo, contenido, resumen, autor, publicada,
imagen, archivo)
    titulo: Mapped[str] = mapped_column(String(255), nullable=False)
    contenido: Mapped[str] = mapped_column(LONGTEXT, nullable=False)  #
Markdown
    resumen: Mapped[str] = mapped_column(String(1000), nullable=False)

    autor: Mapped[str] = mapped_column(String(120), nullable=False,
default="Colegio Médico de Corrientes")
    publicada: Mapped[bool] = mapped_column(Boolean, nullable=False,
default=True)

    imagen: Mapped[Optional[str]] = mapped_column(String(500), nullable=True)
    archivo: Mapped[Optional[str]] = mapped_column(String(500),
nullable=True)

    # Fechas (Node: fechaCreacion, fechaActualizacion)
    fecha_creacion: Mapped[datetime.datetime] = mapped_column(
        DateTime(timezone=True), server_default=func.now(), nullable=False
    )
    fecha_actualizacion: Mapped[datetime.datetime] = mapped_column(
        DateTime(timezone=True), server_default=func.now(),
onupdate=func.now(), nullable=False
```

```
    )

    __table_args__ = (
        Index("ix_noticias_publicada_fecha", "publicada", "fecha_creacion"),
    )
```

---

# 2) Schemas Pydantic

**Archivo nuevo:** `app/schemas/noticias_schema.py`

```python
from datetime import datetime
from typing import Optional
from pydantic import BaseModel, Field

# Lo que recibe el POST (crear)
class NoticiaCreateIn(BaseModel):
    titulo: str
    contenido: str
    resumen: str
    imagen: Optional[str] = None
    archivo: Optional[str] = None
    publicada: Optional[bool] = True
    autor: Optional[str] = None  # si no viene, lo setea el backend

# Lo que recibe el PUT (actualizar)
class NoticiaUpdateIn(BaseModel):
    titulo: Optional[str] = None
    contenido: Optional[str] = None
    resumen: Optional[str] = None
    imagen: Optional[str] = None  # "" para limpiar
    archivo: Optional[str] = None  # "" para limpiar
    publicada: Optional[bool] = None
    autor: Optional[str] = None

# Lo que responde la API (coincide con el front)
class NoticiaOut(BaseModel):
    id: str
    titulo: str
    contenido: str
    resumen: str
    autor: str
    publicada: bool
    fechaCreacion: datetime = Field(..., alias="fecha_creacion")
    fechaActualizacion: datetime = Field(..., alias="fecha_actualizacion")
    imagen: Optional[str] = None
    archivo: Optional[str] = None
```

```
    class Config:
        populate_by_name = True
```

---

# 3) Rutas FastAPI (CRUD Noticias)

**Archivo nuevo:** `app/api/v1/noticias.py`

Replica la semántica del backend Node: - `GET /api/noticias` ⇒ solo **publicadas=true**, orden desc por fecha_creacion - `GET /api/noticias/{id}` - `POST /api/noticias` (requiere login) - `PUT /api/noticias/{id}` (requiere login) - `DELETE /api/noticias/{id}` (requiere login)

```
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy import select, desc
from typing import List

from app.db.database import get_db
from app.db.models import Noticia as NoticiaModel
from app.schemas.noticias_schema import NoticiaCreateIn, NoticiaUpdateIn,
NoticiaOut
from app.auth.deps import get_current_user  # valida el access token

router = APIRouter()

# Helpers de mapeo a la forma que espera el front (id string, alias de
fechas)

def _to_out(row: NoticiaModel) -> NoticiaOut:
    return NoticiaOut(
        id=str(row.id),
        titulo=row.titulo,
        contenido=row.contenido,
        resumen=row.resumen,
        autor=row.autor,
        publicada=row.publicada,
        fecha_creacion=row.fecha_creacion,
        fecha_actualizacion=row.fecha_actualizacion,
        imagen=row.imagen,
        archivo=row.archivo,
    )

@router.get("/", response_model=List[NoticiaOut])
async def listar_noticias(db: AsyncSession = Depends(get_db)):
    q = (
        select(NoticiaModel)
        .where(NoticiaModel.publicada.is_(True))
        .order_by(desc(NoticiaModel.fecha_creacion))
```

```
    )
    res = await db.execute(q)
    rows = res.scalars().all()
    return [_to_out(r) for r in rows]

@router.get("/{id}", response_model=NoticiaOut)
async def obtener_noticia(id: int, db: AsyncSession = Depends(get_db)):
    row = await db.get(NoticiaModel, id)
    if not row:
        raise HTTPException(status_code=404, detail="Noticia no encontrada")
    return _to_out(row)

@router.post("/", response_model=NoticiaOut,
status_code=status.HTTP_201_CREATED)
async def crear_noticia(
    body: NoticiaCreateIn,
    user=Depends(get_current_user),
    db: AsyncSession = Depends(get_db),
):
    # Autor por defecto: del body o genérico
    autor = body.autor or f"Usuario {user.get('nro_socio', 'CMC')}"

    row = NoticiaModel(
        titulo=body.titulo.strip(),
        contenido=body.contenido,
        resumen=body.resumen.strip(),
        autor=autor,
        publicada=bool(body.publicada) if body.publicada is not None else
True,
        imagen=(body.imagen or None) or None,
        archivo=(body.archivo or None) or None,
    )
    db.add(row)
    await db.commit()
    await db.refresh(row)
    return _to_out(row)

@router.put("/{id}")
async def actualizar_noticia(
    id: int,
    body: NoticiaUpdateIn,
    user=Depends(get_current_user),
    db: AsyncSession = Depends(get_db),
):
    row = await db.get(NoticiaModel, id)
    if not row:
        raise HTTPException(status_code=404, detail="Noticia no encontrada")

    if body.titulo is not None:
        row.titulo = body.titulo.strip()
    if body.contenido is not None:
```

```
        row.contenido = body.contenido
    if body.resumen is not None:
        row.resumen = body.resumen.strip()
    if body.autor is not None:
        row.autor = body.autor.strip() or row.autor
    if body.publicada is not None:
        row.publicada = bool(body.publicada)

    # Imagen/archivo: si viene string vacío ⇒ limpiar (NULL)
    if body.imagen is not None:
        row.imagen = body.imagen.strip() or None
    if body.archivo is not None:
        row.archivo = body.archivo.strip() or None

    await db.commit()
    return {"mensaje": "Noticia actualizada exitosamente"}

@router.delete("/{id}")
async def eliminar_noticia(
    id: int,
    user=Depends(get_current_user),
    db: AsyncSession = Depends(get_db),
):
    row = await db.get(NoticiaModel, id)
    if not row:
        raise HTTPException(status_code=404, detail="Noticia no encontrada")
    await db.delete(row)
    await db.commit()
    return {"mensaje": "Noticia eliminada exitosamente"}
```

---

# 4) Registrar el router en tu agregador de rutas

**Archivo:** `app/api/routes.py`

Agregá los imports y el include:

```
from app.api.v1.noticias import router as noticias_router
# ...
api_router.include_router(noticias_router, prefix="/noticias",
tags=["Noticias"])
```

Dejalo similar al resto (medicos, solicitudes, etc.).

---

# 5) Migración Alembic (crear tabla `noticias`)

**Archivo nuevo:** `alembic/versions/20251106_add_noticias_table.py`

Usá exactamente este contenido (ajustá la fecha/ID si preferís). Está preparado para MySQL.

```
from alembic import op
import sqlalchemy as sa
from sqlalchemy.dialects import mysql

# Revisiones
revision = "20251106_add_noticias_table"
down_revision = None  # o la última de tu proyecto, p.ej. "e1dd4b0cf1f5"
branch_labels = None
depends_on = None


def upgrade() -> None:
    op.create_table(
        "noticias",
        sa.Column("id", sa.Integer(), primary_key=True, autoincrement=True),
        sa.Column("titulo", sa.String(length=255), nullable=False),
        sa.Column("contenido", mysql.LONGTEXT, nullable=False),
        sa.Column("resumen", sa.String(length=1000), nullable=False),
        sa.Column("autor", sa.String(length=120), nullable=False),
        sa.Column("publicada", sa.Boolean(), nullable=False,
server_default=sa.text("1")),
        sa.Column("imagen", sa.String(length=500), nullable=True),
        sa.Column("archivo", sa.String(length=500), nullable=True),
        sa.Column("fecha_creacion", sa.DateTime(timezone=True),
nullable=False, server_default=sa.text("CURRENT_TIMESTAMP")),
        sa.Column("fecha_actualizacion", sa.DateTime(timezone=True),
nullable=False, server_default=sa.text("CURRENT_TIMESTAMP")),
        mysql_engine="InnoDB",
        mysql_charset="utf8mb4",
    )
    op.create_index("ix_noticias_publicada_fecha", "noticias", ["publicada",
"fecha_creacion"])


def downgrade() -> None:
    op.drop_index("ix_noticias_publicada_fecha", table_name="noticias")
    op.drop_table("noticias")
```

**Nota:** el `onupdate` lo resolvemos a nivel modelo con `onupdate=func.now()`. Si querés que MySQL lo haga sin pasar por SQLAlchemy, podés reemplazar `fecha_actualizacion` por: `server_default=sa.text("CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP")` (puede variar según versión de MySQL).

# 6) Endpoint de uploads (opcional pero recomendado)

Tu front usa `POST /api/uploads` (con campo `file`) y luego sirve `/uploads/<archivo>`.

**Archivo nuevo:** `app/api/v1/uploads.py`

```
from fastapi import APIRouter, UploadFile, File, Depends, HTTPException
from pathlib import Path
from uuid import uuid4
from app.auth.deps import get_current_user

router = APIRouter()
UPLOAD_DIR = Path("uploads")
UPLOAD_DIR.mkdir(parents=True, exist_ok=True)

@router.post("/")
async def upload(file: UploadFile = File(...),
user=Depends(get_current_user)):
    # nombre seguro y extensión original
    ext = (Path(file.filename).suffix or "").lower()
    name = f"{uuid4().hex}{ext}"
    dest = UPLOAD_DIR / name

    data = await file.read()
    dest.write_bytes(data)

    return {
        "url": f"/uploads/{name}",
        "mimetype": file.content_type,
        "filename": name,
    }
```

**Montar estáticos:** `app/main.py`

```
from fastapi.staticfiles import StaticFiles

# ... después de crear app = FastAPI(...)
app.mount("/uploads", StaticFiles(directory="uploads"), name="uploads")
```

**Registrar el router:** `app/api/routes.py`

```
from app.api.v1.uploads import router as uploads_router
# ...
```

```
api_router.include_router(uploads_router, prefix="/uploads",
tags=["Uploads"])
```

---

# 7) Variables y compatibilidad con el front actual

- La **forma** del objeto `NoticiaOut` **coincide** con tu front (`id: string`, `fechaCreacion`, `fechaActualizacion`, etc.).
- `GET /api/noticias` devuelve **solo publicadas**, igual que en Node.
- `POST/PUT/DELETE` requieren login (como `authenticateToken` en Node). Uso `get_current_user` (ajustá a `require_scope` si querés roles específicos).
- **Autor**: si el front no envía `autor`, se completa con `Usuario <nro_socio>` tomado del token. Podés cambiar a email/ nombre si ya lo tenés disponible en tus claims.

---

# 8) Pasos para aplicar

1. **Crear archivos** indicados arriba y pegar los contenidos.
2. **Agregar** el modelo en `models.py` y los includes en `routes.py` / `main.py`.
3. **Migración**:
4. Si generás con Alembic: `alembic revision -m "add noticias table"` y reemplazá el archivo generado por el contenido que te dejé.
5. Aplicá: `alembic upgrade head`.
6. **Probar**:
7. `GET /api/noticias` (debería devolver `[]`).
8. Crear con `POST /api/noticias` (token requerido).
9. Ver detalle `GET /api/noticias/{id}`.
10. Subir imagen/pdf con `POST /api/uploads` y usar la `url` resultante en `imagen` / `archivo`.

---

# 9) Notes sobre performance/índices

- Índice compuesto `publicada, fecha_creacion` para la home/listado.
- `contenido` en `LONGTEXT` permite Markdown largo sin problemas.

---

# 10) ¿Qué mapeé del backend Node?

- **Campos**: `titulo, contenido (md), resumen, autor, publicada, imagen, archivo, fechaCreacion/Actualizacion`.
- **Rutas**: mismas rutas y semántica (`list` solo publicadas, CRUD autenticado).
- **Uploads**: endpoint compatible (`url`, `mimetype`, `filename`) y exposición estática en `/uploads/*`.

Si querés, después integramos permisos finos (p.ej. `scope: "noticias:editar"`) o agregamos **borrador** (publicada=false) visible para admins. También puedo adaptarlo a **slugs** legibles si lo necesitás.