**Cambios concretos (sin repetir funcionalidades)**

**1) Sustituye el middleware “todo-en-uno”**

**Acción**: Deja de usar app/middleware/security\_middleware.py. Divide responsabilidades:

**a) Headers seguros tipo Helmet**

app/middleware/security\_headers.py

from starlette.middleware.base import BaseHTTPMiddleware

from fastapi import Request

class SecurityHeadersMiddleware(BaseHTTPMiddleware):

async def dispatch(self, request: Request, call\_next):

resp = await call\_next(request)

resp.headers["X-Content-Type-Options"] = "nosniff"

resp.headers["X-Frame-Options"] = "DENY"

resp.headers["Referrer-Policy"] = "strict-origin-when-cross-origin"

resp.headers["Strict-Transport-Security"] = "max-age=31536000; includeSubDomains"

resp.headers["Permissions-Policy"] = "geolocation=(), microphone=()"

if request.url.path == "/docs":

resp.headers["Content-Security-Policy"] = (

"default-src 'self' https://cdn.jsdelivr.net; "

"style-src 'self' https://cdn.jsdelivr.net 'unsafe-inline'; "

"script-src 'self' https://cdn.jsdelivr.net 'unsafe-inline'; "

"img-src 'self' https://cdn.jsdelivr.net data:; "

"font-src 'self' https://cdn.jsdelivr.net;"

)

else:

resp.headers["Content-Security-Policy"] = "default-src 'self'"

return resp

**b) Rate limiting global (SlowAPI)**

app/security/rate\_limiting.py

from slowapi import Limiter

from slowapi.util import get\_remote\_address

import os

DEFAULT\_LIMIT = os.getenv("RATE\_LIMIT", "60/minute")

limiter = Limiter(key\_func=get\_remote\_address, default\_limits=[DEFAULT\_LIMIT])

def limit(rule: str | None = None):

return limiter.limit(rule or DEFAULT\_LIMIT)

Sí. Aquí tienes **qué cambiar en tu repo** para el numeral 1, con detalle y sin duplicados. Incluyo un **patch** listo para git apply.

**Opción A — Patch listo**

**Descarga:** numeral1\_security\_split.patch

Aplicas desde la raíz que contiene la carpeta backend/:

git checkout -b chore/security-middlewares-split

git apply --3way --reject /ruta/a/numeral1\_security\_split.patch

git add .

git commit -m "Split security middleware: headers + SlowAPI rate limiting; remove legacy SecurityMiddleware"

El patch hace:

* **Crea** app/middleware/security\_headers.py
* **Crea** app/security/rate\_limiting.py
* **Edita** app/main.py para **quitar** SecurityMiddleware y **añadir** SlowAPI + headers
* **Elimina** app/middleware/security\_middleware.py

**Opción B — Pasos manuales**

**1) Eliminar el “todo-en-uno”**

* Borra el archivo: app/middleware/security\_middleware.py.
* Quita en app/main.py:
* from app.middleware.security\_middleware import SecurityMiddleware
* app.add\_middleware(SecurityMiddleware)

**2) Crear headers tipo Helmet**

**Archivo nuevo:** app/middleware/security\_headers.py

from starlette.middleware.base import BaseHTTPMiddleware

from fastapi import Request

class SecurityHeadersMiddleware(BaseHTTPMiddleware):

async def dispatch(self, request: Request, call\_next):

resp = await call\_next(request)

resp.headers["X-Content-Type-Options"] = "nosniff"

resp.headers["X-Frame-Options"] = "DENY"

resp.headers["Referrer-Policy"] = "strict-origin-when-cross-origin"

resp.headers["Strict-Transport-Security"] = "max-age=31536000; includeSubDomains"

resp.headers["Permissions-Policy"] = "geolocation=(), microphone=()"

if request.url.path == "/docs":

resp.headers["Content-Security-Policy"] = (

"default-src 'self' https://cdn.jsdelivr.net; "

"style-src 'self' https://cdn.jsdelivr.net 'unsafe-inline'; "

"script-src 'self' https://cdn.jsdelivr.net 'unsafe-inline'; "

"img-src 'self' https://cdn.jsdelivr.net data:; "

"font-src 'self' https://cdn.jsdelivr.net;"

)

else:

resp.headers["Content-Security-Policy"] = "default-src 'self'"

return resp

**3) Rate limiting global con SlowAPI**

**Instala:**

pip install slowapi

**Archivo nuevo:** app/security/rate\_limiting.py

from slowapi import Limiter

from slowapi.util import get\_remote\_address

import os

DEFAULT\_LIMIT = os.getenv("RATE\_LIMIT", "60/minute")

limiter = Limiter(key\_func=get\_remote\_address, default\_limits=[DEFAULT\_LIMIT])

def limit(rule: str | None = None):

return limiter.limit(rule or DEFAULT\_LIMIT)

**Var de entorno opcional:**

# Windows

setx RATE\_LIMIT "60/minute"

# Linux/Mac

export RATE\_LIMIT="60/minute"

**4) Conectar en app/main.py**

Añade imports:

from slowapi.errors import RateLimitExceeded

from slowapi.middleware import SlowAPIMiddleware

from app.middleware.security\_headers import SecurityHeadersMiddleware

from app.security.rate\_limiting import limiter

Tras crear app = FastAPI(...), agrega en este **orden**:

# Seguridad HTTP

app.add\_middleware(SecurityHeadersMiddleware)

# Rate limit global

app.state.limiter = limiter

app.add\_exception\_handler(RateLimitExceeded, lambda r, e: e.response)

app.add\_middleware(SlowAPIMiddleware)

No vuelvas a añadir un rate-limiter propio en memoria. SlowAPI ya lo cubre.

**5) Verificación**

* Headers:
* curl -i http://localhost:8000/docs | grep -E "Content-Security-Policy|X-Frame-Options|X-Content-Type-Options"
* Límite:  
  Llama >60 veces en un minuto a un endpoint público. Debe devolver 429.

¿Seguimos con el **numeral 2** (sanitización de input y validación central) cuando confirmes estos cambios?

**2) Autenticación JWT central (sin repetir “current\_user”)**

app/security/authentication.py

from \_\_future\_\_ import annotations

from typing import Callable, Awaitable

from starlette.middleware.base import BaseHTTPMiddleware

from fastapi import Request, HTTPException, status, Depends

from fastapi.security import OAuth2PasswordBearer

from sqlalchemy.ext.asyncio import AsyncSession

from sqlalchemy import select

from app.core.jwt import decode\_token

from app.core.security import get\_async\_db

from app.models.user import User

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="/auth/token")

PUBLIC\_PATHS: tuple[str, ...] = ("/openapi.json","/docs","/docs/oauth2-redirect","/redoc","/health","/auth/login","/auth/token")

class JWTAuthMiddleware(BaseHTTPMiddleware):

async def dispatch(self, request: Request, call\_next: Callable[[Request], Awaitable]):

p = request.url.path

if p in PUBLIC\_PATHS or p.startswith("/public/") or (p == "/" and request.method == "GET"):

return await call\_next(request)

auth = request.headers.get("Authorization","")

parts = auth.split()

if len(parts)!=2 or parts[0].lower()!="bearer":

raise HTTPException(status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Missing or invalid Authorization header")

payload = decode\_token(parts[1], expected\_type="access")

if not payload or "sub" not in payload:

raise HTTPException(status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token")

request.state.user\_id = payload["sub"]

request.state.scopes = payload.get("scopes", [])

return await call\_next(request)

async def get\_current\_user(request: Request, db: AsyncSession = Depends(get\_async\_db)) -> User:

user\_id = getattr(request.state, "user\_id", None)

if not user\_id:

token = await oauth2\_scheme(request)

payload = decode\_token(token, expected\_type="access")

if not payload or "sub" not in payload:

raise HTTPException(status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or missing token")

user\_id = payload["sub"]

user = (await db.execute(select(User).where(User.id==user\_id))).scalar\_one\_or\_none()

if not user or not user.active:

raise HTTPException(status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Inactive or missing user")

return user

**Elimina duplicidad en dependencies/current\_user.py**

**Opción estable sin romper imports existentes**: convertirlo en *fachada*.  
app/dependencies/current\_user.py

from \_\_future\_\_ import annotations

from fastapi import Depends

from app.security.authentication import get\_current\_user as \_get

from app.models.user import User

async def get\_current\_user(user: User = Depends(\_get)) -> User:

return user

Y borra su require\_scope viejo (lo reemplaza requires\_scopes abajo).

**3) RBAC declarativo central**

app/security/authorization.py

from \_\_future\_\_ import annotations

from fastapi import Depends, HTTPException, status

from sqlalchemy.ext.asyncio import AsyncSession

from sqlalchemy import select

from app.models.user import User

from app.models.role import RoleType, Role

from app.core.security import get\_async\_db

from app.security.authentication import get\_current\_user

def \_deny(): raise HTTPException(status\_code=status.HTTP\_403\_FORBIDDEN, detail="Access denied")

def has\_permission(user: User, scope: str) -> bool:

if user.superuser: return True

if not user.role: return False

scopes = user.role.scopes or []

return scope in scopes

def requires\_role(\*accepted: RoleType):

async def \_dep(user: User = Depends(get\_current\_user), db: AsyncSession = Depends(get\_async\_db)):

if user.superuser: return

if not user.role\_id: \_deny()

role\_type = (await db.execute(select(Role.role\_type).where(Role.id==user.role\_id))).scalar\_one\_or\_none()

if role\_type not in accepted: \_deny()

return \_dep

def requires\_scopes(\*required\_scopes: str):

async def \_dep(user: User = Depends(get\_current\_user)):

if user.superuser: return

for s in required\_scopes:

if not has\_permission(user, s): \_deny()

return \_dep

**4) Sanitización central del body JSON (XSS básico)**

app/security/input\_validation.py

from \_\_future\_\_ import annotations

import json, re, html

from typing import Any, Mapping

from starlette.middleware.base import BaseHTTPMiddleware

from fastapi import Request, HTTPException, status, UploadFile

SCRIPT\_RE = re.compile(r"<\s\*\/?\s\*script[^>]\*>", re.I)

EVENT\_HANDLER\_RE = re.compile(r"on\w+\s\*=", re.I)

JS\_URL\_RE = re.compile(r"javascript\s\*:", re.I)

def \_sanitize\_str(s: str, max\_len: int | None = None) -> str:

if max\_len is not None: s = s[:max\_len]

s = SCRIPT\_RE.sub("", s); s = EVENT\_HANDLER\_RE.sub("", s); s = JS\_URL\_RE.sub("", s)

return html.escape(s, quote=True)

def \_walk\_and\_clean(obj: Any, field\_max: Mapping[str,int] | None = None) -> Any:

if isinstance(obj, dict):

out = {}

for k,v in obj.items():

out[k] = \_walk\_and\_clean(v, field\_max)

if isinstance(v, str) and field\_max and k in field\_max:

out[k] = \_sanitize\_str(v, max\_len=field\_max[k])

return out

if isinstance(obj, list): return [\_walk\_and\_clean(v, field\_max) for v in obj]

if isinstance(obj, str): return \_sanitize\_str(obj)

return obj

class BodySanitizationMiddleware(BaseHTTPMiddleware):

def \_\_init\_\_(self, app, field\_max: Mapping[str,int] | None = None):

super().\_\_init\_\_(app); self.field\_max = field\_max or {}

async def dispatch(self, request: Request, call\_next):

if request.headers.get("content-type","").lower().startswith("application/json"):

raw = await request.body()

try: data = json.loads(raw.decode("utf-8"))

except Exception: raise HTTPException(status\_code=status.HTTP\_400\_BAD\_REQUEST, detail="Invalid JSON")

cleaned = \_walk\_and\_clean(data, self.field\_max)

new\_body = json.dumps(cleaned).encode("utf-8")

async def receive(): return {"type":"http.request","body":new\_body,"more\_body":False}

request.\_receive = receive # type: ignore

return await call\_next(request)

ALLOWED\_CONTENT\_TYPES = {"image/png","image/jpeg","application/pdf"}

MAX\_UPLOAD\_BYTES = 5 \* 1024 \* 1024

def validate\_upload(file: UploadFile):

if file.content\_type not in ALLOWED\_CONTENT\_TYPES:

raise HTTPException(status\_code=400, detail="Invalid file type")

**5) Logging estructurado y handlers globales**

Reemplaza utils/logging\_config.py por logging estructurado en **un** módulo.

app/core/logging.py

import logging, logging.config, os, sys, json

from app.core.config import settings

class JsonFormatter(logging.Formatter):

def format(self, r: logging.LogRecord) -> str:

payload = {"level": r.levelname, "logger": r.name, "msg": r.getMessage()}

for k in ("user\_id","ip","path","method"):

if hasattr(r,k): payload[k] = getattr(r,k)

return json.dumps(payload, ensure\_ascii=False)

def setup\_logging() -> None:

if settings.APP\_ENV == "production":

fmt = {"()": JsonFormatter}

handlers = {"stdout":{"class":"logging.StreamHandler","stream":"ext://sys.stdout","formatter":"json","level":"INFO"}}

root = {"level":"INFO","handlers":["stdout"]}

else:

fmt = {"console":{"format":"%(levelname)s | %(name)s | %(message)s"}}

handlers = {"console":{"class":"logging.StreamHandler","stream":"ext://sys.stdout","formatter":"console","level":"DEBUG"}}

root = {"level":"INFO","handlers":["console"]}

logging.config.dictConfig({

"version":1,"disable\_existing\_loggers":False,

"formatters":fmt,"handlers":handlers,"root":root,

"loggers":{"security":{"level":"INFO","handlers":root["handlers"],"propagate":False},

"uvicorn.access":{"level":"WARNING"},

"uvicorn.error":{"level":"INFO"},

"sqlalchemy.engine.Engine":{"level":"WARNING"}}

})

app/core/errors.py

from fastapi import Request, HTTPException

from fastapi.responses import JSONResponse

from fastapi.exceptions import RequestValidationError

from app.core.config import settings

import logging

log = logging.getLogger("security")

def install\_exception\_handlers(app):

@app.exception\_handler(HTTPException)

async def http\_exc(request: Request, exc: HTTPException):

if exc.status\_code >= 500:

log.error("http\_error", extra={"path": str(request.url), "method": request.method, "status": exc.status\_code})

return JSONResponse({"detail": exc.detail}, status\_code=exc.status\_code)

@app.exception\_handler(RequestValidationError)

async def validation\_exc(request: Request, exc: RequestValidationError):

log.warning("validation\_error", extra={"path": str(request.url), "method": request.method})

if settings.APP\_ENV == "production":

return JSONResponse({"detail": "Invalid request"}, status\_code=422)

return JSONResponse({"detail": exc.errors()}, status\_code=422)

@app.exception\_handler(Exception)

async def unhandled\_exc(request: Request, exc: Exception):

log.error("unhandled\_error", extra={"path": str(request.url), "method": request.method})

msg = "Internal server error" if settings.APP\_ENV == "production" else str(exc)

return JSONResponse({"detail": msg}, status\_code=500)

**6) Utilidades sin duplicar**

**Sustituye** tu app/utils/security\_utils.py por esta versión consolidada:

from \_\_future\_\_ import annotations

import re, html

from urllib.parse import urlparse

from typing import Any, Dict

SQL\_INJECTION\_PATTERNS = [

r"(\b(SELECT|INSERT|UPDATE|DELETE|DROP|UNION|EXEC|ALTER|CREATE|TRUNCATE)\b)",

r"(\-\-|\#|\/\\*)",

r"(\b(OR|AND)\s+['\"]?[01]['\"]?\s\*[=<>])",

r"(;|\|&)",

]

SENSITIVE\_FIELDS = {"password","token","secret","authorization","api\_key"}

def sanitize\_text(value: str, \*, max\_len: int | None = None) -> str:

if max\_len is not None: value = value[:max\_len]

value = re.sub(r"<\s\*\/?\s\*script[^>]\*>", "", value, flags=re.I)

value = re.sub(r"on\w+\s\*=", "", value, flags=re.I)

value = re.sub(r"javascript\s\*:", "", value, flags=re.I)

return html.escape(value, quote=True)

def sanitize\_dict(d: Dict[str, Any], field\_max: Dict[str,int] | None = None) -> Dict[str, Any]:

out: Dict[str, Any] = {}

for k,v in d.items():

if isinstance(v, str): out[k] = sanitize\_text(v, max\_len=(field\_max or {}).get(k))

elif isinstance(v, dict): out[k] = sanitize\_dict(v, field\_max)

elif isinstance(v, list): out[k] = [sanitize\_text(x) if isinstance(x, str) else x for x in v]

else: out[k] = v

return out

def looks\_like\_sql\_injection(text: str) -> bool:

return any(re.search(p, text, flags=re.I) for p in SQL\_INJECTION\_PATTERNS)

def obfuscate\_sensitive(d: Dict[str, Any]) -> Dict[str, Any]:

return {k: ("\*\*\*REDACTED\*\*\*" if k.lower() in SENSITIVE\_FIELDS else v) for k,v in d.items()}

def is\_safe\_redirect(url: str, allowed\_hosts: list[str] | None = None) -> bool:

allowed\_hosts = allowed\_hosts or []

p = urlparse(url)

if not p.netloc: # relativo

return True

return p.netloc.lower() in [h.lower() for h in allowed\_hosts]

**7) Cambios en imports para no repetir**

* app/main.py
  + Quita: from app.middleware.security\_middleware import SecurityMiddleware
  + Quita uso de SecurityMiddleware.
  + Cambia logging:  
    **antes** from app.utils.logging\_config import setup\_logging  
    **ahora** from app.core.logging import setup\_logging
  + Añade:
  + from slowapi.errors import RateLimitExceeded
  + from slowapi.middleware import SlowAPIMiddleware
  + from app.middleware.security\_headers import SecurityHeadersMiddleware
  + from app.security.input\_validation import BodySanitizationMiddleware
  + from app.security.authentication import JWTAuthMiddleware
  + from app.security.rate\_limiting import limiter
  + from app.core.errors import install\_exception\_handlers
  + setup\_logging()
  + # CORS ya con core/cors.py
  + app.add\_middleware(SecurityHeadersMiddleware)
  + app.add\_middleware(BodySanitizationMiddleware) # opcional: BodySanitizationMiddleware(app, field\_max={"name":100})
  + app.add\_middleware(JWTAuthMiddleware)
  + app.state.limiter = limiter
  + app.add\_exception\_handler(RateLimitExceeded, lambda r, e: e.response)
  + app.add\_middleware(SlowAPIMiddleware)
  + install\_exception\_handlers(app)
* app/routers/auth.py  
  Cambia import de current user si lo usas:
* # de:
* # from app.dependencies.current\_user import get\_current\_user
* # a:
* from app.security.authentication import get\_current\_user
* app/routers/user.py  
  Quita import bcrypt. Usa:
* from app.core.security import get\_password\_hash, verify\_password

y **reemplaza** cualquier uso directo de bcrypt por esas funciones.

* **Mantén** core/jwt.py como única fuente de decode\_token/create\_access\_token.

**8) Variables de entorno**

* JWT\_SECRET\_KEY, JWT\_ALGORITHM, JWT\_ACCESS\_TOKEN\_EXPIRE\_MINUTES
* ALLOWED\_ORIGINS
* APP\_ENV = production o development
* RATE\_LIMIT ejemplo 60/minute

**9) Dependencias**

pip install "python-jose[cryptography]" slowapi

**Análisis de riesgos mitigados**

* **Auth central**: tokens inválidos/expirados bloqueados antes del endpoint.
* **RBAC**: permisos y roles en un solo lugar. Menos errores por endpoints sin protección.
* **XSS básico en input**: sanitización del body JSON.
* **Open Redirect**: helper para validar URLs.
* **CORS restrictivo**: ya en core/cors.py.
* **Headers de seguridad**: HSTS, no-sniff, anti-framing, CSP.
* **Rate limiting**: por IP con SlowAPI.
* **Errores**: sin stack traces en prod, logging listo para SIEM.
* **Hashing**: bcrypt centralizado en core/security.py.

Diagnóstico rápido de tu ZIP (backend/app):

* CORS seguro: ✅ (core/cors.py, ALLOWED\_ORIGINS leído).
* JWT + middleware: ✅ (security/authentication.py en main.py).
* Headers de seguridad y CSP: ✅ (middleware/security\_headers.py en main.py).
* Rate limiting global (SlowAPI): ✅ (en main.py).
* Sanitización JSON body: ✅ (BodySanitizationMiddleware en main.py).
* Manejadores globales de errores: ✅ (core/errors.py instalado).
* Logging estructurado: ✅ (core/logging.py usado).
* ORM parametrizado: ✅ (no hallé text() ni SQL crudo).
* Subidas de archivo: ❌ sin validación de tipo/tamaño en routers.
* RBAC definido: ✅ (security/authorization.py) pero **no aplicado** en routers.
* .env en repo: ❌ sin .gitignore para excluir .env\*.
* Rate-limit por endpoint sensible (login): ❌
* Auditoría de eventos de seguridad: parcial (solo en core/errors.py).

Abajo tienes los **pasos concretos** con código para copiar.

**1) Aplicar RBAC en routers críticos**

Protege creación/edición/borrado. Ejemplo en routers/user.py y routers/product.py:

# al inicio

from fastapi import Depends

from app.security.authorization import requires\_role, requires\_scopes

from app.models.role import RoleType

# a nivel de router para todo el módulo (ejemplo solo lectura)

router = APIRouter(prefix="/users", tags=["Users"],

dependencies=[Depends(requires\_scopes("users:read"))])

# en endpoints sensibles añade dependencias específicas

@router.post("/", dependencies=[Depends(requires\_role(RoleType.ADMIN)),

Depends(requires\_scopes("users:create"))])

async def create\_user\_endpoint(...):

...

@router.delete("/{user\_id}",

dependencies=[Depends(requires\_role(RoleType.ADMIN)),

Depends(requires\_scopes("users:delete"))])

async def delete\_user\_endpoint(...):

...

Repite en product.py, purchase.py, entry.py, role.py, document.py, etc.

**2) Validar tipo/tamaño en uploads**

Añade validación antes de leer el archivo. Reutiliza el helper y limita tamaño.

# en cada router con UploadFile

from fastapi import UploadFile, File, HTTPException

from app.security.input\_validation import ALLOWED\_CONTENT\_TYPES

import io

MAX\_UPLOAD\_BYTES = 5 \* 1024 \* 1024 # 5 MiB

def \_ensure\_valid\_upload(file: UploadFile, max\_bytes: int = MAX\_UPLOAD\_BYTES):

if file.content\_type not in ALLOWED\_CONTENT\_TYPES:

raise HTTPException(status\_code=400, detail="Invalid file type")

# lee como stream limitado

chunk = b""

buf = io.BytesIO()

while True:

chunk = file.file.read(1024 \* 64)

if not chunk:

break

buf.write(chunk)

if buf.tell() > max\_bytes:

raise HTTPException(status\_code=400, detail="File too large")

buf.seek(0)

return buf

@router.post("/import", status\_code=201)

async def import\_users(file: UploadFile = File(...), ...):

data\_io = \_ensure\_valid\_upload(file)

text = data\_io.read().decode("utf-8", errors="ignore")

...

Aplícalo en: routers/user.py, catalog\_router.py, entry.py, purchase.py.

**3) Limitar campos por longitud desde el middleware**

Activa límites coherentes en main.py:

from app.security.input\_validation import BodySanitizationMiddleware

app.add\_middleware(

BodySanitizationMiddleware,

field\_max={"code":10,"name":100,"description":500,"basis":20}

)

**4) Endurecer CSP**

Amplía middleware/security\_headers.py:

# dentro dispatch(), antes del return

# Para /docs conservas jsdelivr pero añade directivas fuertes:

if request.url.path == "/docs":

csp = (

"default-src 'self' https://cdn.jsdelivr.net; "

"style-src 'self' https://cdn.jsdelivr.net 'unsafe-inline'; "

"script-src 'self' https://cdn.jsdelivr.net 'unsafe-inline'; "

"img-src 'self' https://cdn.jsdelivr.net data:; "

"font-src 'self' https://cdn.jsdelivr.net; "

"object-src 'none'; base-uri 'self'; frame-ancestors 'none'; form-action 'self'"

)

else:

csp = "default-src 'self'; object-src 'none'; base-uri 'self'; frame-ancestors 'none'; form-action 'self'"

response.headers["Content-Security-Policy"] = csp

# Además añade:

response.headers["Cache-Control"] = "no-store" # en respuestas sensibles (token)

Para /auth/token y /auth/login puedes establecer Cache-Control: no-store en la respuesta del endpoint.

**5) Rate-limit más estricto en login**

Aplica límite dedicado al login para frenar fuerza bruta.

# routers/auth.py

from app.security.rate\_limiting import limit

@router.post("/token", dependencies=[limit("5/minute")], response\_model=Token)

async def oauth2\_token(...):

...

**6) Log de eventos de seguridad clave**

Añade logs **estructurados** en autenticación/autorización.

# routers/auth.py

import logging

logsec = logging.getLogger("security")

# en credenciales inválidas

logsec.warning("login\_failed", extra={"ip": request.client.host, "path": "/auth/token", "method": "POST"})

# en login ok

logsec.info("login\_success", extra={"user\_id": str(user.id), "ip": request.client.host, "path": "/auth/token", "method": "POST"})

# security/authorization.py (en \_deny)

import logging

\_logsec = logging.getLogger("security")

def \_deny():

\_logsec.warning("access\_denied", extra={"path":"<auto>", "method":"<auto>"})

raise HTTPException(status\_code=403, detail="Access denied")

Opcional: incluye user\_id si lo tienes en contexto.

**7) Bloqueo por intentos fallidos (simple)**

Mitiga fuerza bruta además del rate limit. Implementación in-memory:

# routers/auth.py (arriba del módulo)

from time import time

FAILED: dict[str, list[float]] = {}

WINDOW, MAX\_FAILS, BLOCK\_SEC = 300, 5, 900 # 5 min, 5 intentos, 15 min de bloqueo

def is\_blocked(ip: str) -> bool:

now = time()

fails = [t for t in FAILED.get(ip, []) if now - t < WINDOW]

if len(fails) >= MAX\_FAILS:

# si último fallo fue hace < BLOCK\_SEC => bloqueado

return now - fails[-1] < BLOCK\_SEC

return False

def add\_fail(ip: str):

now = time()

lst = FAILED.get(ip, [])

lst = [t for t in lst if now - t < WINDOW] + [now]

FAILED[ip] = lst

@router.post("/token", response\_model=Token)

async def oauth2\_token(..., request: Request):

ip = request.client.host

if is\_blocked(ip):

raise HTTPException(status\_code=429, detail="Too many failed attempts")

...

if not user or not await verify\_password(password, user.password):

add\_fail(ip)

raise HTTPException(status\_code=401, detail="Credenciales de usuario inválidas",

headers={"WWW-Authenticate": "Bearer"})

En producción usa Redis para persistencia.

**8) No commitear secretos**

Crea .gitignore en la raíz del repo:

# Env y credenciales

.env

.env.\*

!.env.example

# Artefactos Python

\_\_pycache\_\_/

\*.pyc

\*.pyo

\*.pyd

.venv/

venv/

Copia tu .env.dev a .env.example **sin** secretos reales como plantilla.

**9) Usar límites de importación declarados en .env**

Estás definiendo MAX\_IMPORT\_FILE\_SIZE\_MB y MAX\_IMPORT\_ROWS pero no se usan. Léalos en core/config.py y aplícalos en los importadores:

# core/config.py (Settings)

MAX\_IMPORT\_FILE\_SIZE\_MB: int = 5

MAX\_IMPORT\_ROWS: int = 1000

# en routers/\* que importan CSV

from app.core.config import settings

MAX\_BYTES = settings.MAX\_IMPORT\_FILE\_SIZE\_MB \* 1024 \* 1024

MAX\_ROWS = settings.MAX\_IMPORT\_ROWS

# tras validar tamaño, corta filas

rows = 0

for row in reader:

rows += 1

if rows > MAX\_ROWS:

raise HTTPException(status\_code=400, detail="Demasiadas filas en el archivo")

**10) Ajustes JWT recomendados**

Ya validas iss, aud, exp, iat, nbf. Completa con:

* **Binding opcional**: guardar jti y lista de revocados en Redis para “logout”.
* **Refresh rotation**: emitir nuevo refresh en /refresh y revocar el usado.
* **Scopes**: estándar scope en token ya lo devuelves; úsalo en requires\_scopes.

**11) Producción**

* HSTS solo si sirves por HTTPS real.
* Define ALLOWED\_ORIGINS con tu dominio exacto, nunca \*.
* Limita tamaño de petición en el **reverse proxy**: Nginx client\_max\_body\_size 5m;.

**Resultado esperado**

Con estos cambios cubres: control de acceso efectivo por endpoint (RBAC), validación de uploads, límites de fuerza bruta, logs de seguridad accionables, CSP robusto, y protección de secretos. Si quieres, te preparo parches por archivo para aplicar con git apply.

Aquí tienes los **pasos exactos** con **rutas** y **cambios**. Copia y pega.

# 1) Escapar comodines en búsquedas ILIKE

**Archivo:** app/utils/security\_utils.py  
**Acción:** agregar helper.

# === agregar al final del archivo ===

def escape\_like(term: str) -> str:

term = term.replace("\\", "\\\\").replace("%", "\\%").replace("\_", "\\\_")

return term

**Archivos a modificar (patrón de búsqueda con search):**

* app/crud/payment\_term.py
* app/crud/brand.py
* app/crud/category.py
* app/crud/product.py
* (y cualquier CRUD que use .ilike(f"%{search}%"))

**Acción en cada CRUD:** reemplazar el patrón “ilike con f-string” por versión segura.

# antes

like = f"%{search}%"

query = query.where( Model.name.ilike(like) | Model.code.ilike(like) )

# después

from sqlalchemy import or\_

from app.utils.security\_utils import escape\_like

pattern = f"%{escape\_like(search)}%"

query = query.where(

or\_(

Model.name.ilike(pattern, escape="\\"),

Model.code.ilike(pattern, escape="\\"),

)

)

Repite para todos los campos buscables (name, code, description, etc.).

# 2) Validación de uploads (tipo y tamaño)

**Archivo:** app/security/input\_validation.py  
**Acción:** crear o **reemplazar** validate\_upload.

# === reemplazar/crear ===

from app.core.config import settings

from fastapi import HTTPException, UploadFile

import io

ALLOWED\_CONTENT\_TYPES = {"image/png", "image/jpeg", "application/pdf"}

def validate\_upload(file: UploadFile) -> bytes:

if file.content\_type not in ALLOWED\_CONTENT\_TYPES:

raise HTTPException(status\_code=400, detail="Invalid file type")

max\_bytes = settings.MAX\_IMPORT\_FILE\_SIZE\_MB \* 1024 \* 1024

buf = io.BytesIO()

while True:

chunk = file.file.read(65536)

if not chunk:

break

buf.write(chunk)

if buf.tell() > max\_bytes:

raise HTTPException(status\_code=400, detail="File too large")

return buf.getvalue()

**Archivos a modificar (tienen UploadFile):**

* app/routers/user.py
* app/routers/catalog\_router.py
* app/routers/entry.py
* app/routers/purchase.py

**Acción en cada endpoint que lee archivo:** usar validate\_upload.

# antes

content = await file.read()

text = content.decode("utf-8", errors="ignore")

# después

from app.security.input\_validation import validate\_upload

raw\_bytes = validate\_upload(file)

text = raw\_bytes.decode("utf-8", errors="ignore")

Si no es texto, procesa raw\_bytes como corresponda. Aplica también límites de **filas** si importas CSV (ver paso 5).

# 3) Límites por campo en sanitización (XSS de entrada)

**Archivo:** app/main.py  
**Acción:** asegurar field\_max.

from app.security.input\_validation import BodySanitizationMiddleware

app.add\_middleware(

BodySanitizationMiddleware,

field\_max={"code":10, "name":100, "description":500, "basis":20}

)

# 4) CSP más estricta y no almacenar respuestas sensibles

**Archivo:** app/middleware/security\_headers.py  
**Acción:** reforzar CSP y no-store en /auth.

# dentro de dispatch(), antes del return

if request.url.path == "/docs":

csp = (

"default-src 'self' https://cdn.jsdelivr.net; "

"style-src 'self' https://cdn.jsdelivr.net 'unsafe-inline'; "

"script-src 'self' https://cdn.jsdelivr.net 'unsafe-inline'; "

"img-src 'self' https://cdn.jsdelivr.net data:; "

"font-src 'self' https://cdn.jsdelivr.net; "

"object-src 'none'; base-uri 'self'; frame-ancestors 'none'; form-action 'self'"

)

else:

csp = "default-src 'self'; object-src 'none'; base-uri 'self'; frame-ancestors 'none'; form-action 'self'"

response.headers["Content-Security-Policy"] = csp

# no-store en rutas de autenticación

if request.url.path.startswith("/auth"):

response.headers["Cache-Control"] = "no-store"

# 5) Límite de filas en importaciones CSV

**Archivos:** routers que importan CSV (user.py, catalog\_router.py, entry.py, purchase.py)  
**Acción:** usar settings.MAX\_IMPORT\_ROWS.

from app.core.config import settings

rows = 0

for row in reader:

rows += 1

if rows > settings.MAX\_IMPORT\_ROWS:

raise HTTPException(status\_code=400, detail="Demasiadas filas en el archivo")

# si guardas strings, puedes sanitizarlas:

# from app.utils.security\_utils import sanitize\_text

# name = sanitize\_text(row.get("name",""), max\_len=100)

# 6) RBAC efectivo en endpoints de escritura

**Archivos típicos:**

* app/routers/user.py, product.py, purchase.py, entry.py, role.py, document.py, brand.py, etc.  
  **Acción:** añadir dependencias por endpoint.

from fastapi import Depends

from app.security.authorization import requires\_role, requires\_scopes

from app.models.role import RoleType

@router.post("/", dependencies=[Depends(requires\_role(RoleType.ADMIN)),

Depends(requires\_scopes("users:create"))])

async def create\_user\_endpoint(...): ...

@router.delete("/{user\_id}", dependencies=[Depends(requires\_role(RoleType.ADMIN)),

Depends(requires\_scopes("users:delete"))])

async def delete\_user\_endpoint(...): ...

Ajusta users:\*, products:\*, entries:\*, purchases:\*, etc., según tu modelo de permisos.

# 7) Rate limit estricto en login

**Archivo:** app/routers/auth.py  
**Acción:** limitar /token y loggear eventos.

from app.security.rate\_limiting import limit

import logging

logsec = logging.getLogger("security")

@router.post("/token", response\_model=Token, dependencies=[limit("5/minute")])

async def oauth2\_token(..., request: Request):

# en fallo

logsec.warning("login\_failed", extra={"ip": request.client.host, "path": "/auth/token", "method": "POST"})

# en éxito

logsec.info("login\_success", extra={"user\_id": str(user.id), "ip": request.client.host, "path": "/auth/token", "method": "POST"})

...

# 8) Prohibir SQL crudo (política de revisión)

**Acción:** busca y evita estas firmas en todo el repo:

* sqlalchemy.text(
* .execute(f" … )`
* .execute("SELECT ... %s" % var)

Si alguna es inevitable, **parametriza** siempre y **revisa** manualmente.

# 9) Verificaciones rápidas (post-cambio)

* CORS:
* curl -s -D - -o /dev/null -H "Origin: http://localhost:3000" http://localhost:8000/docs \
* | tr -d '\r' | grep -i access-control-allow-origin
* Headers:
* curl -I http://localhost:8000/docs | grep -iE 'content-security-policy|x-frame-options|x-content-type-options'
* Login rate limit: 6+ requests seguidas a /auth/token ⇒ **429**.
* Upload inválido: tipo no permitido o > límite ⇒ **400**.
* Búsquedas: probando \_ o % en search no debe romper ni ampliar coincidencias inintencionales.

Eso cubre **SQLi** (comodines escapados y política anti-SQL crudo) y **XSS** (entrada, uploads, y salida reforzada con CSP), además de permisos y límites.

Principio del formulario