Implementación de Ciberseguridad Reutilizable para Backend FastAPI

Voy a implementar un sistema de ciberseguridad modular y reutilizable para tu backend. Te guiaré paso a paso en la implementación.

🛡️ 1. Middleware de Seguridad Centralizado

**Ubicación:**backend/app/middleware/security\_middleware.py

python

from fastapi import Request, HTTPException

from fastapi.responses import JSONResponse

from starlette.middleware.base import BaseHTTPMiddleware

import time

import logging

import re

from typing import Callable, Dict, Any

from app.core.config import settings

logger = logging.getLogger(\_\_name\_\_)

class SecurityMiddleware(BaseHTTPMiddleware):

def \_\_init\_\_(self, app):

super().\_\_init\_\_(app)

self.rate\_limiter: Dict[str, list] = {}

self.blocked\_ips: Dict[str, float] = {}

self.rate\_limit = getattr(settings, 'RATE\_LIMIT', 100)

self.burst\_limit = getattr(settings, 'BURST\_LIMIT', 15)

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

client\_ip = request.client.host

*# 1. Verificar IP bloqueada*

if self.\_is\_ip\_blocked(client\_ip):

return JSONResponse(

status\_code=429,

content={"detail": "Demasiadas solicitudes. IP temporalmente bloqueada."}

)

*# 2. Rate limiting*

if not self.\_check\_rate\_limit(client\_ip):

return JSONResponse(

status\_code=429,

content={"detail": "Demasiadas solicitudes"}

)

*# 3. Validar headers de seguridad*

security\_errors = self.\_validate\_request\_headers(request)

if security\_errors:

return JSONResponse(

status\_code=400,

content={"detail": security\_errors}

)

*# 4. Procesar solicitud*

response = await call\_next(request)

*# 5. Agregar headers de seguridad a la respuesta*

self.\_add\_security\_headers(response)

return response

def \_check\_rate\_limit(self, ip: str) -> bool:

current\_time = time.time()

window\_start = current\_time - 60 *# Ventana de 1 minuto*

*# Limpiar registros antiguos*

self.rate\_limiter[ip] = [t for t in self.rate\_limiter.get(ip, []) if t > window\_start]

if len(self.rate\_limiter[ip]) >= self.rate\_limit:

if len(self.rate\_limiter[ip]) >= self.rate\_limit + self.burst\_limit:

self.blocked\_ips[ip] = current\_time + 300 *# Bloquear por 5 minutos*

return False

self.rate\_limiter.setdefault(ip, []).append(current\_time)

return True

def \_is\_ip\_blocked(self, ip: str) -> bool:

block\_time = self.blocked\_ips.get(ip, 0)

if block\_time > time.time():

return True

elif ip in self.blocked\_ips:

del self.blocked\_ips[ip]

return False

def \_validate\_request\_headers(self, request: Request) -> str:

"""Valida headers de seguridad en la solicitud"""

errors = []

*# Validar Content-Type para requests con body*

if request.method in ['POST', 'PUT', 'PATCH']:

content\_type = request.headers.get('content-type', '')

if not content\_type.startswith('application/json'):

errors.append("Content-Type debe ser application/json")

*# Validar User-Agent*

user\_agent = request.headers.get('user-agent', '')

if not user\_agent or len(user\_agent) > 256:

errors.append("User-Agent inválido")

return "; ".join(errors) if errors else ""

def \_add\_security\_headers(self, response):

security\_headers = {

"X-Content-Type-Options": "nosniff",

"X-Frame-Options": "DENY",

"X-XSS-Protection": "1; mode=block",

"Strict-Transport-Security": "max-age=31536000; includeSubDomains",

"Content-Security-Policy": "default-src 'self'",

"Referrer-Policy": "strict-origin-when-cross-origin",

"Permissions-Policy": "geolocation=(), microphone=()"

}

for header, value in security\_headers.items():

response.headers[header] = value

🛡️ 2. Utilidades de Seguridad Reutilizables

**Ubicación:**backend/app/utils/security\_utils.py

python

import re

import html

from fastapi import HTTPException, status

from typing import Any, Dict, Optional

import logging

logger = logging.getLogger(\_\_name\_\_)

class SecurityUtils:

*# Patrones de inyección SQL*

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"(;|\|&)",

]

*# Patrones de XSS*

XSS\_PATTERNS = [

r"<script.\*?>.\*?</script>",

r"javascript:",

r"onerror\s\*=",

r"onload\s\*=",

r"onclick\s\*=",

r"vbscript:",

r"expression\s\*\(",

]

*# Campos sensibles que deben ser ofuscados en logs*

SENSITIVE\_FIELDS = {

'password', 'token', 'secret', 'api\_key', 'credit\_card',

'cvv', 'expiration\_date', 'ssn', 'phone\_number', 'email'

}

@classmethod

def sanitize\_input(cls, input\_string: str, field\_name: str = None) -> str:

"""

Sanitiza y valida una entrada de texto

"""

if not isinstance(input\_string, str):

return input\_string

*# 1. Validar longitud máxima*

if field\_name and hasattr(cls, 'FIELD\_MAX\_LENGTHS'):

max\_length = cls.FIELD\_MAX\_LENGTHS.get(field\_name, 255)

if len(input\_string) > max\_length:

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail=f"{field\_name} excede la longitud máxima permitida"

)

*# 2. Filtrar inyección SQL*

for pattern in cls.SQL\_INJECTION\_PATTERNS:

if re.search(pattern, input\_string, re.IGNORECASE):

logger.warning(f"Intento de inyección SQL detectado en campo {field\_name}: {input\_string}")

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail="Entrada contiene patrones potencialmente peligrosos"

)

*# 3. Sanitizar XSS*

cleaned = input\_string

for pattern in cls.XSS\_PATTERNS:

cleaned = re.sub(pattern, "", cleaned, flags=re.IGNORECASE)

*# 4. Escapar caracteres HTML*

cleaned = html.escape(cleaned)

return cleaned

@classmethod

def sanitize\_data(cls, data: Dict[str, Any]) -> Dict[str, Any]:

"""

Sanitiza todos los campos de un diccionario

"""

sanitized = {}

for key, value in data.items():

if isinstance(value, str):

sanitized[key] = cls.sanitize\_input(value, key)

elif isinstance(value, dict):

sanitized[key] = cls.sanitize\_data(value)

elif isinstance(value, list):

sanitized[key] = [cls.sanitize\_data(item) if isinstance(item, dict) else

cls.sanitize\_input(item, key) if isinstance(item, str) else item

for item in value]

else:

sanitized[key] = value

return sanitized

@classmethod

def obfuscate\_sensitive\_data(cls, data: Dict[str, Any]) -> Dict[str, Any]:

"""

Ofusca datos sensibles para logging

"""

obfuscated = data.copy()

for key in obfuscated.keys():

if any(sensitive in key.lower() for sensitive in cls.SENSITIVE\_FIELDS):

obfuscated[key] = "\*\*\*REDACTED\*\*\*"

return obfuscated

*# Configuración de longitudes máximas por tipo de campo*

SecurityUtils.FIELD\_MAX\_LENGTHS = {

'code': 10,

'name': 100,

'description': 500,

'basis': 20,

'email': 254,

'username': 50,

'password': 128,

}

🛡️ 3. Mejoras al Sistema de Autenticación JWT

**Ubicación:**backend/app/core/security\_enhanced.py

python

from datetime import datetime, timedelta

from typing import Optional, List

from jose import JWTError, jwt

from fastapi import Depends, HTTPException, status

from fastapi.security import OAuth2PasswordBearer

from sqlalchemy.ext.asyncio import AsyncSession

from app.core.config import settings

from app.db.session import get\_db

from app.models.user import User

from app.crud.user import get\_user\_by\_id

import logging

logger = logging.getLogger(\_\_name\_\_)

*# Extender el esquema OAuth2 para incluir scopes*

class OAuth2PasswordBearerWithScopes(OAuth2PasswordBearer):

def \_\_init\_\_(self, tokenUrl: str, scopes: List[str] = None):

super().\_\_init\_\_(tokenUrl=tokenUrl)

self.required\_scopes = scopes or []

oauth2\_scheme = OAuth2PasswordBearerWithScopes(tokenUrl="auth/token")

class TokenBlacklist:

def \_\_init\_\_(self):

self.revoked\_tokens = set()

def add\_token(self, token: str, expire\_time: int):

self.revoked\_tokens.add((token, expire\_time))

def is\_revoked(self, token: str) -> bool:

current\_time = datetime.utcnow().timestamp()

*# Limpiar tokens expirados*

self.revoked\_tokens = {(t, exp) for t, exp in self.revoked\_tokens if exp > current\_time}

return any(t == token for t, \_ in self.revoked\_tokens)

token\_blacklist = TokenBlacklist()

def create\_access\_token(data: dict, expires\_delta: Optional[timedelta] = None, scopes: List[str] = None):

to\_encode = data.copy()

if expires\_delta:

expire = datetime.utcnow() + expires\_delta

else:

expire = datetime.utcnow() + timedelta(minutes=settings.JWT\_ACCESS\_TOKEN\_EXPIRE\_MINUTES)

to\_encode.update({

"exp": expire,

"type": "access",

"scopes": scopes or ["read"] *# Scopes por defecto*

})

return jwt.encode(to\_encode, settings.JWT\_SECRET\_KEY, algorithm=settings.JWT\_ALGORITHM)

async def get\_current\_user(

token: str = Depends(oauth2\_scheme),

db: AsyncSession = Depends(get\_db)

) -> User:

credentials\_exception = HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Could not validate credentials",

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

)

try:

payload = jwt.decode(token, settings.JWT\_SECRET\_KEY, algorithms=[settings.JWT\_ALGORITHM])

user\_id: str = payload.get("sub")

token\_type: str = payload.get("type")

if user\_id is None or token\_type != "access":

raise credentials\_exception

except JWTError:

raise credentials\_exception

*# Verificar si el token está revocado*

if token\_blacklist.is\_revoked(token):

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Token ha sido revocado"

)

user = await get\_user\_by\_id(db, user\_id)

if user is None or not user.active:

raise credentials\_exception

return user

def require\_scope(required\_scope: str):

"""Dependency para verificar scopes del token"""

def scope\_checker(token: str = Depends(oauth2\_scheme)):

try:

payload = jwt.decode(token, settings.JWT\_SECRET\_KEY, algorithms=[settings.JWT\_ALGORITHM])

token\_scopes = payload.get("scopes", [])

if required\_scope not in token\_scopes:

raise HTTPException(

status\_code=status.HTTP\_403\_FORBIDDEN,

detail="Permisos insuficientes"

)

except JWTError:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Token inválido"

)

return token

return scope\_checker

🛡️ 4. Modelo Base Seguro para Pydantic

**Ubicación:**backend/app/schemas/security\_schemas.py

python

from pydantic import BaseModel, Field, field\_validator

from typing import Optional, Any

from app.utils.security\_utils import SecurityUtils

class SecureBaseModel(BaseModel):

"""

Modelo base Pydantic con validaciones de seguridad integradas

"""

@field\_validator('\*', mode='before')

@classmethod

def validate\_all\_fields(cls, value: Any, info):

"""Validador que aplica sanitización a todos los campos string"""

field\_name = info.field\_name

if isinstance(value, str):

return SecurityUtils.sanitize\_input(value, field\_name)

*# Para diccionarios y listas, aplicar sanitización recursiva*

if isinstance(value, dict):

return SecurityUtils.sanitize\_data(value)

elif isinstance(value, list):

return [SecurityUtils.sanitize\_data(item) if isinstance(item, dict) else

SecurityUtils.sanitize\_input(item, field\_name) if isinstance(item, str) else item

for item in value]

return value

class Config:

arbitrary\_types\_allowed = True

validate\_assignment = True

🛡️ 5. CRUD Base Seguro

**Ubicación:**backend/app/crud/base\_security.py

python

from typing import Type, TypeVar, Generic, Optional, Dict, Any

from uuid import UUID

from sqlalchemy.ext.asyncio import AsyncSession

from sqlalchemy.future import select

from sqlalchemy.exc import SQLAlchemyError, IntegrityError

from fastapi import HTTPException, status

import logging

from app.utils.security\_utils import SecurityUtils

T = TypeVar('T')

logger = logging.getLogger(\_\_name\_\_)

class CRUDSecurityBase(Generic[T]):

def \_\_init\_\_(self, model: Type[T]):

self.model = model

async def create\_with\_security(

self,

db: AsyncSession,

obj\_in: Dict[str, Any],

user\_id: UUID,

additional\_checks: Optional[callable] = None

) -> T:

"""

Crear objeto con validaciones de seguridad

"""

try:

*# 1. Sanitizar datos de entrada*

sanitized\_data = SecurityUtils.sanitize\_data(obj\_in)

*# 2. Validaciones adicionales personalizadas*

if additional\_checks:

await additional\_checks(db, sanitized\_data)

*# 3. Verificar unicidad si el modelo tiene campo 'code'*

if hasattr(self.model, 'code') and 'code' in sanitized\_data:

existing = await db.execute(

select(self.model).where(

getattr(self.model, 'code') == sanitized\_data['code']

)

)

if existing.scalar\_one\_or\_none():

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail=f"{self.model.\_\_name\_\_} con este código ya existe"

)

*# 4. Crear objeto*

db\_obj = self.model(\*\*sanitized\_data)

if hasattr(db\_obj, 'user\_id'):

db\_obj.user\_id = user\_id

db.add(db\_obj)

await db.flush()

return db\_obj

except IntegrityError as e:

await db.rollback()

logger.error(f"Error de integridad al crear {self.model.\_\_name\_\_}: {str(e)}")

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail="Error de integridad de datos"

)

except HTTPException:

await db.rollback()

raise

except Exception as e:

await db.rollback()

logger.error(f"Error inesperado al crear {self.model.\_\_name\_\_}: {str(e)}")

raise HTTPException(

status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR,

detail="Error interno del servidor"

)

async def update\_with\_security(

self,

db: AsyncSession,

obj\_id: UUID,

obj\_in: Dict[str, Any],

user\_id: UUID

) -> Optional[T]:

"""

Actualizar objeto con validaciones de seguridad

"""

try:

*# 1. Sanitizar datos de entrada*

sanitized\_data = SecurityUtils.sanitize\_data(obj\_in)

*# 2. Obtener objeto existente*

result = await db.execute(

select(self.model).where(getattr(self.model, 'id') == obj\_id)

)

db\_obj = result.scalar\_one\_or\_none()

if not db\_obj:

return None

*# 3. Aplicar cambios*

for field, value in sanitized\_data.items():

if hasattr(db\_obj, field):

setattr(db\_obj, field, value)

*# 4. Actualizar timestamp si existe*

if hasattr(db\_obj, 'updated\_at'):

from datetime import datetime

db\_obj.updated\_at = datetime.utcnow()

await db.flush()

return db\_obj

except SQLAlchemyError as e:

await db.rollback()

logger.error(f"Error de base de datos al actualizar {self.model.\_\_name\_\_}: {str(e)}")

raise HTTPException(

status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR,

detail="Error en la base de datos"

)

🛡️ 6. Configuración de CORS Segura

**Ubicación:**backend/app/core/cors.py

python

from fastapi.middleware.cors import CORSMiddleware

from app.core.config import settings

def setup\_secure\_cors(app):

"""

Configuración segura de CORS

"""

*# Obtener origenes permitidos desde settings*

allowed\_origins = getattr(settings, 'ALLOWED\_ORIGINS', '').split(',')

app.add\_middleware(

CORSMiddleware,

allow\_origins=allowed\_origins,

allow\_credentials=True,

allow\_methods=["GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"],

allow\_headers=[

"Authorization",

"Content-Type",

"X-Requested-With",

"X-CSRF-Token"

],

expose\_headers=["Content-Range", "X-Total-Count"],

max\_age=600, *# 10 minutos para preflight requests*

)

🛡️ 7. Integración en la Aplicación Principal

**Modificación de**backend/app/main.py**:**

python

from fastapi import FastAPI

from app.middleware.security\_middleware import SecurityMiddleware

from app.core.cors import setup\_secure\_cors

from app.core.config import settings

import logging

*# Configuración de logging*

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

app = FastAPI(

title="API Segura",

description="API con medidas de seguridad implementadas",

version="1.0.0",

docs\_url="/docs" if settings.APP\_ENV == "development" else None,

redoc\_url=None,

)

*# 1. Aplicar middleware de seguridad*

app.add\_middleware(SecurityMiddleware)

*# 2. Configurar CORS seguro*

setup\_secure\_cors(app)

*# 3. Importar routers*

from app.routers import payment\_terms, users, auth

app.include\_router(auth.router, prefix="/auth", tags=["Authentication"])

app.include\_router(users.router, prefix="/users", tags=["Users"])

app.include\_router(payment\_terms.router, prefix="/api/paymentterms", tags=["PaymentTerms"])

@app.get("/health")

async def health\_check():

"""Endpoint de health check sin autenticación"""

return {"status": "healthy", "security": "enabled"}

🛡️ 8. Actualización del CRUD de PaymentTerm

**Modificación de**backend/app/crud/payment\_term.py**:**

python

from app.crud.base\_security import CRUDSecurityBase

from app.models.payment\_term import PaymentTerm

class CRUDPaymentTerm(CRUDSecurityBase[PaymentTerm]):

def \_\_init\_\_(self):

super().\_\_init\_\_(PaymentTerm)

async def create\_payment\_term(self, db: AsyncSession, payment\_term\_data: dict, user\_id: UUID):

"""Usar el método seguro del base"""

return await self.create\_with\_security(db, payment\_term\_data, user\_id)

*# Instancia global*

crud\_payment\_term = CRUDPaymentTerm()

*# Reemplazar la función original con:*

async def create\_payment\_term(db: AsyncSession, payment\_term\_data: dict, user\_id: UUID):

return await crud\_payment\_term.create\_payment\_term(db, payment\_term\_data, user\_id)

🛡️ 9. Actualización de Schemas

**Modificación de**backend/app/schemas/payment\_term.py**:**

python

from app.schemas.security\_schemas import SecureBaseModel

class PaymentTermBase(SecureBaseModel):

*# Los campos permanecen iguales pero ahora heredan de SecureBaseModel*

code: str = Field(..., min\_length=1, max\_length=10)

name: str = Field(..., min\_length=1, max\_length=100)

*# ... resto de campos*

🛡️ 10. Configuración de Variables de Entorno Mejorada

**Modificación de**backend/app/core/config.py**:**

python

class Settings(BaseSettings):

*# ... configuraciones existentes ...*

*# Seguridad*

RATE\_LIMIT: int = 100

BURST\_LIMIT: int = 15

ALLOWED\_ORIGINS: str = "http://localhost:3000,https://tudominio.com"

*# Archivos*

MAX\_IMPORT\_FILE\_SIZE\_MB: int = 10

MAX\_IMPORT\_ROWS: int = 1000

class Config:

env\_file = ".env.dev"

📋 Pasos de Implementación:

1. **Crear los nuevos archivos** de seguridad en las ubicaciones indicadas
2. **Actualizar los archivos existentes** con las modificaciones mostradas
3. **Ejecutar pruebas** para verificar que la funcionalidad existente siga trabajando
4. **Configurar las variables de entorno** adicionales en .env.dev
5. **Probar las medidas de seguridad** intentando realizar ataques básicos

🔍 Explicación de Cada Elemento:

1. **Middleware de Seguridad**

* **Rate Limiting**: Previene ataques de fuerza bruta
* **IP Blocking**: Bloquea IPs maliciosas temporalmente
* **Header Validation**: Valida headers de entrada
* **Security Headers**: Agrega headers de seguridad HTTP

2. **Utilidades de Seguridad**

* **SQL Injection Prevention**: Filtra patrones de inyección SQL
* **XSS Protection**: Sanitiza entradas para prevenir cross-site scripting
* **Data Obfuscation**: Ofusca datos sensibles en logs
* **Input Validation**: Valida longitudes y formatos

3. **Autenticación Mejorada**

* **Token Revocation**: Permite revocar tokens JWT
* **Scope-based Authorization**: Control de acceso basado en scopes
* **Enhanced Token Security**: Mejoras en la gestión de tokens

4. **Modelos Base Seguros**

* **Automatic Sanitization**: Sanitización automática de todos los campos
* **Input Validation**: Validación integrada en los modelos Pydantic
* **Reusable**: Puede ser heredado por todos los schemas

5. **CRUD Base Seguro**

* **Centralized Security**: Lógica de seguridad reutilizable
* **Automatic Sanitization**: Sanitización automática de datos
* **Error Handling**: Manejo consistente de errores
* **Extensible**: Fácil de extender para nuevos modelos

6. **CORS Seguro**

* **Origin Restrictions**: Solo origenes permitidos
* **Header Control**: Headers específicos permitidos
* **Preflight Caching**: Optimización de requests preflight

Esta implementación proporciona una base sólida de seguridad que es completamente reutilizable para todos los modelos y endpoints de tu aplicación FastAPI.