AT5 Audit Platform: Documentacion Tecnica Arquitectonica

Tesis Doctoral en Ingenieria de Software

Plataforma de Auditoria Interactiva para Verificacion y Validacion de Software

---

Autor: Sistema AT5 MCP **Version**: 1.0.0 **Fecha**: Enero 2026 **Clasificacion**: Documento Tecnico de Nivel Doctoral
Estandares de Referencia: ISO/IEC/IEEE 29119, ISTQB, OWASP, IEEE 730

Resumen Ejecutivo

La presente documentacion describe la arquitectura tecnica completa de **AT5 Audit Platform**, un sistema empresarial de auditoria de software disenado bajo los principios de la ingenieria de software moderna y los estandares internacionales de testing. El sistema implementa un enfoque de verificacion y validacion (V&V) que cumple con ISO/IEC/IEEE 29119-3 para documentacion de pruebas, incorporando mecanismos de integridad criptografica basados en cadenas de hash SHA-256 similares a blockchain, firmas digitales con certificados verificables, y un modelo de control de acceso basado en roles (RBAC) de cinco niveles.

Indice de Contenidos

---

1. Introduccion y Contexto Teorico

1.1 Fundamentacion del Problema

La auditoria de software representa un proceso critico en el ciclo de vida del desarrollo de software (SDLC), definido por el estandar IEEE 1028-2008 como "un examen independiente de un producto de software, proceso de software, o conjunto de procesos de software realizado por una tercera parte para evaluar el cumplimiento con especificaciones, estandares, acuerdos contractuales, u otros criterios".

La complejidad de los sistemas de software modernos, combinada con requisitos regulatorios cada vez mas estrictos (SOX, HIPAA, GDPR, ISO 27001), ha creado una necesidad imperativa de herramientas de auditoria que no solo documenten los resultados de las pruebas, sino que proporcionen **evidencia inmutable** y **verificable criptograficamente** de todo el proceso de auditoria.

1.2 Objetivos del Sistema

AT5 Audit Platform fue disenado con los siguientes objetivos arquitectonicos:

1. **Trazabilidad Completa**: Cada accion en el sistema genera un registro de auditoria con hash criptografico encadenado
2. **Integridad de Datos**: Implementacion de firma digital con certificados SHA-256 para garantizar no-repudio
3. **Cumplimiento Normativo**: Adherencia estricta a ISO/IEC/IEEE 29119 para estructura de planes y casos de prueba
4. **Escalabilidad Horizontal**: Arquitectura desacoplada que permite escalamiento independiente de componentes
5. **Experiencia de Usuario Moderna**: Interfaz reactiva construida con React Server Components y streaming

1.3 Stack Tecnologico

```
┌─────────────────────────────────────────────────────────────┐
│                    CAPA DE PRESENTACION                     │
├─────────────────────────────────────────────────────────────┤
│  Next.js 15 (App Router) │ React 19 │ TypeScript 5.x        │
│  Tailwind CSS 3.4 │ Shadcn/UI │ Recharts 2.x │ Lucide Icons  │
├─────────────────────────────────────────────────────────────┤
│                    CAPA DE APLICACION                       │
├─────────────────────────────────────────────────────────────┤
│  NextAuth.js v5 │ Zod Validation │ Server Actions           │
│  API Routes (Route Handlers) │ Middleware Authentication    │
├─────────────────────────────────────────────────────────────┤
│                    CAPA DE PERSISTENCIA                     │
├─────────────────────────────────────────────────────────────┤
│  Prisma ORM 5.x │ SQLite (dev) / PostgreSQL (prod)          │
│  bcrypt.js (Password Hashing) │ crypto (SHA-256)            │
└─────────────────────────────────────────────────────────────┘
```

---

2. Arquitectura del Sistema

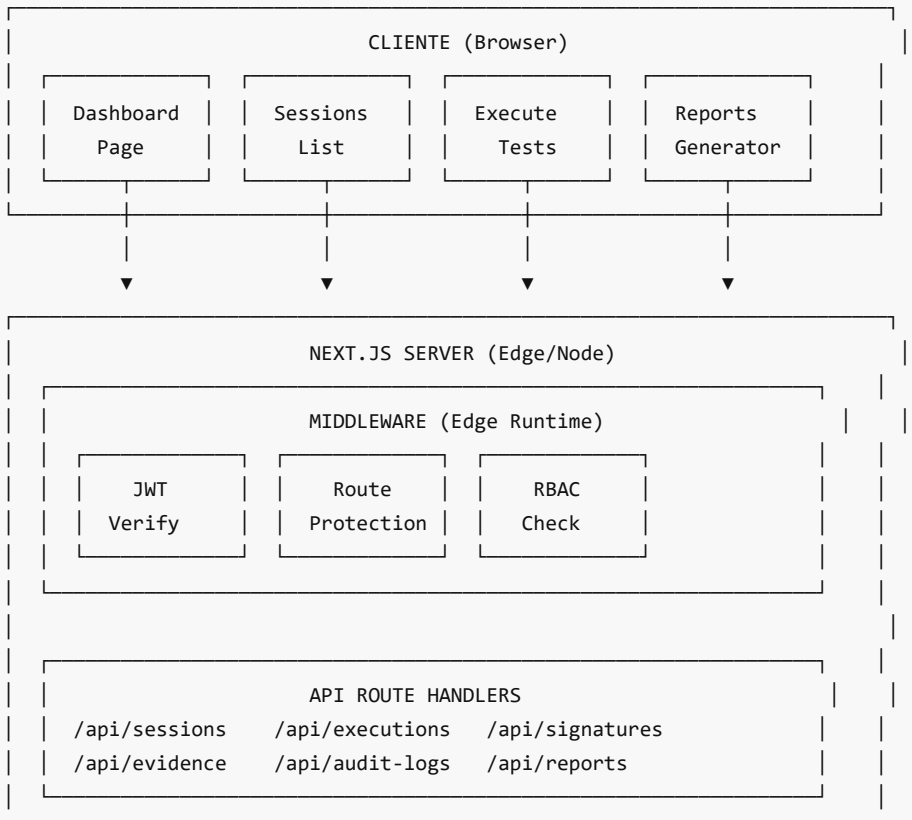
2.1 Patron Arquitectonico: Layered Architecture con Domain-Driven Design

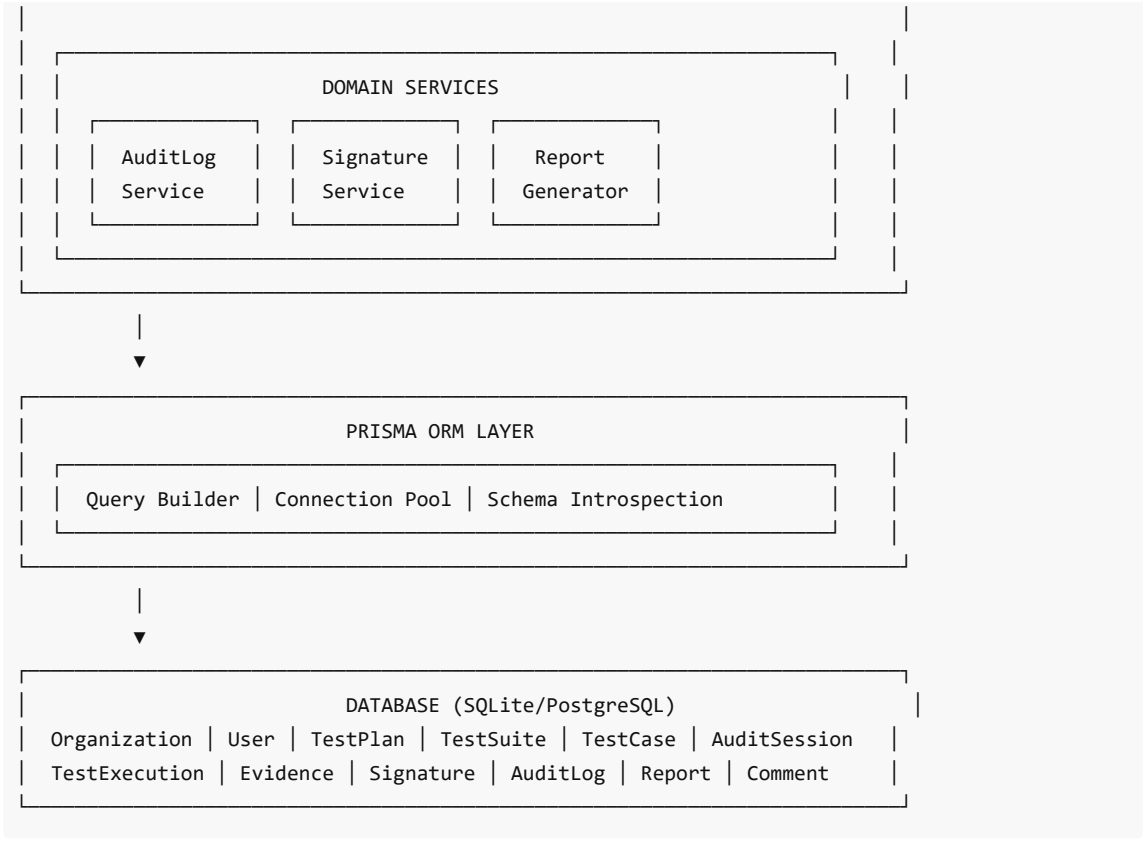
El sistema implementa una arquitectura en capas con separacion clara de responsabilidades, siguiendo los principios de Domain-Driven Design (DDD) propuestos por Eric Evans (2003):

```
at5-audit-platform/
├── src/
│   ├── app/                    # Capa de Presentacion (Next.js App Router)
│   │   ├── (dashboard)/        # Grupo de rutas autenticadas
│   │   │   ├── dashboard/      # Vista principal
│   │   │   ├── sessions/       # Gestion de sesiones de auditoria
│   │   │   │   ├── [id]/       # Detalle de sesion (ruta dinamica)
│   │   │   │   │   ├── execute/ # Ejecucion interactiva de pruebas
│   │   │   │   │   └── page.tsx # Vista de sesion individual
│   │   │   │   ├── new/        # Creacion de nueva sesion
│   │   │   │   └── page.tsx    # Lista de sesiones
│   │   │   ├── audit-log/      # Visualizacion de logs de auditoria
│   │   │   ├── reports/        # Generacion y descarga de reportes
│   │   │   ├── metrics/        # Metricas y KPIs
```

```
│   │   │   └── settings/       # Configuracion del sistema
│   │   ├── api/                # Capa de API (Route Handlers)
│   │   │   ├── auth/           # Endpoints de autenticacion
│   │   │   ├── sessions/       # CRUD de sesiones
│   │   │   ├── executions/     # Ejecucion de pruebas
│   │   │   ├── evidence/       # Gestion de evidencias
│   │   │   ├── signatures/     # Firmas digitales
│   │   │   ├── audit-logs/     # Consulta de logs
│   │   │   └── reports/        # Generacion de reportes
│   │   ├── login/              # Pagina de autenticacion
│   │   └── layout.tsx          # Layout raiz con providers
│   ├── components/             # Componentes reutilizables
│   │   ├── ui/                 # Primitivos de UI (Shadcn)
│   │   └── dashboard/          # Componentes especificos del dashboard
│   └── lib/                    # Capa de Dominio y Servicios
│       ├── prisma.ts           # Cliente Prisma singleton
│       ├── auth.ts             # Configuracion NextAuth
│       ├── audit-log.ts        # Servicio de audit logging
│       ├── signature.ts        # Servicio de firmas digitales
│       └── validations/        # Esquemas Zod
├── prisma/
│   ├── schema.prisma           # Definicion del modelo de datos
│   └── seed.ts                 # Datos iniciales
└── public/                     # Assets estaticos
```

2.2 Diagrama de Componentes



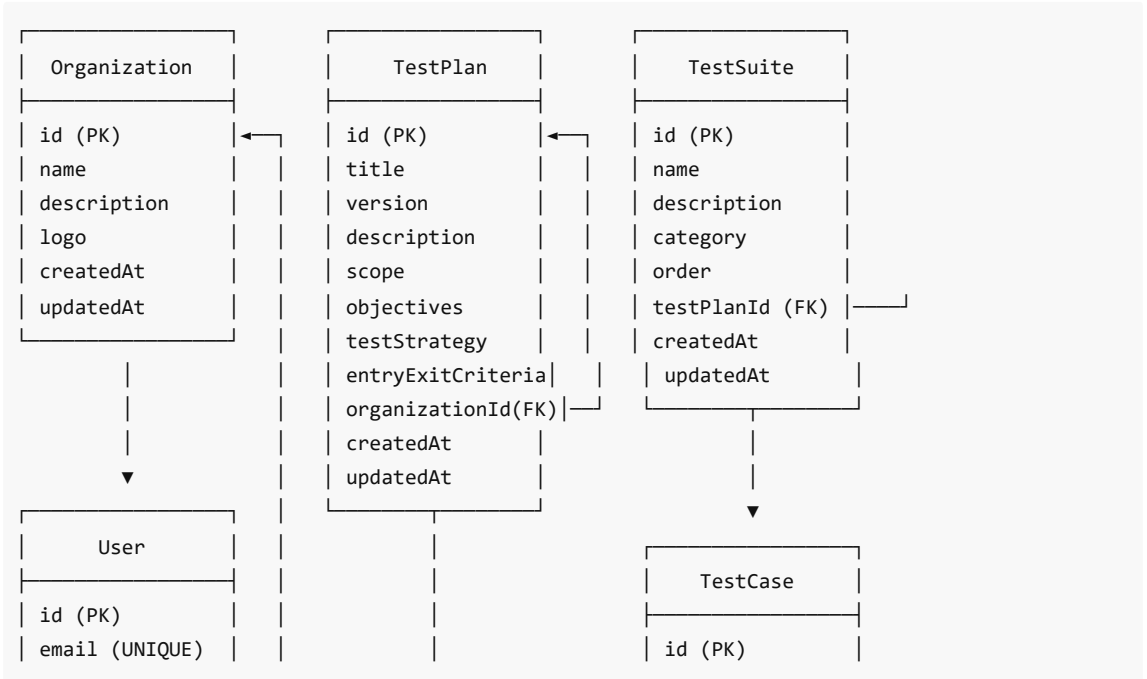


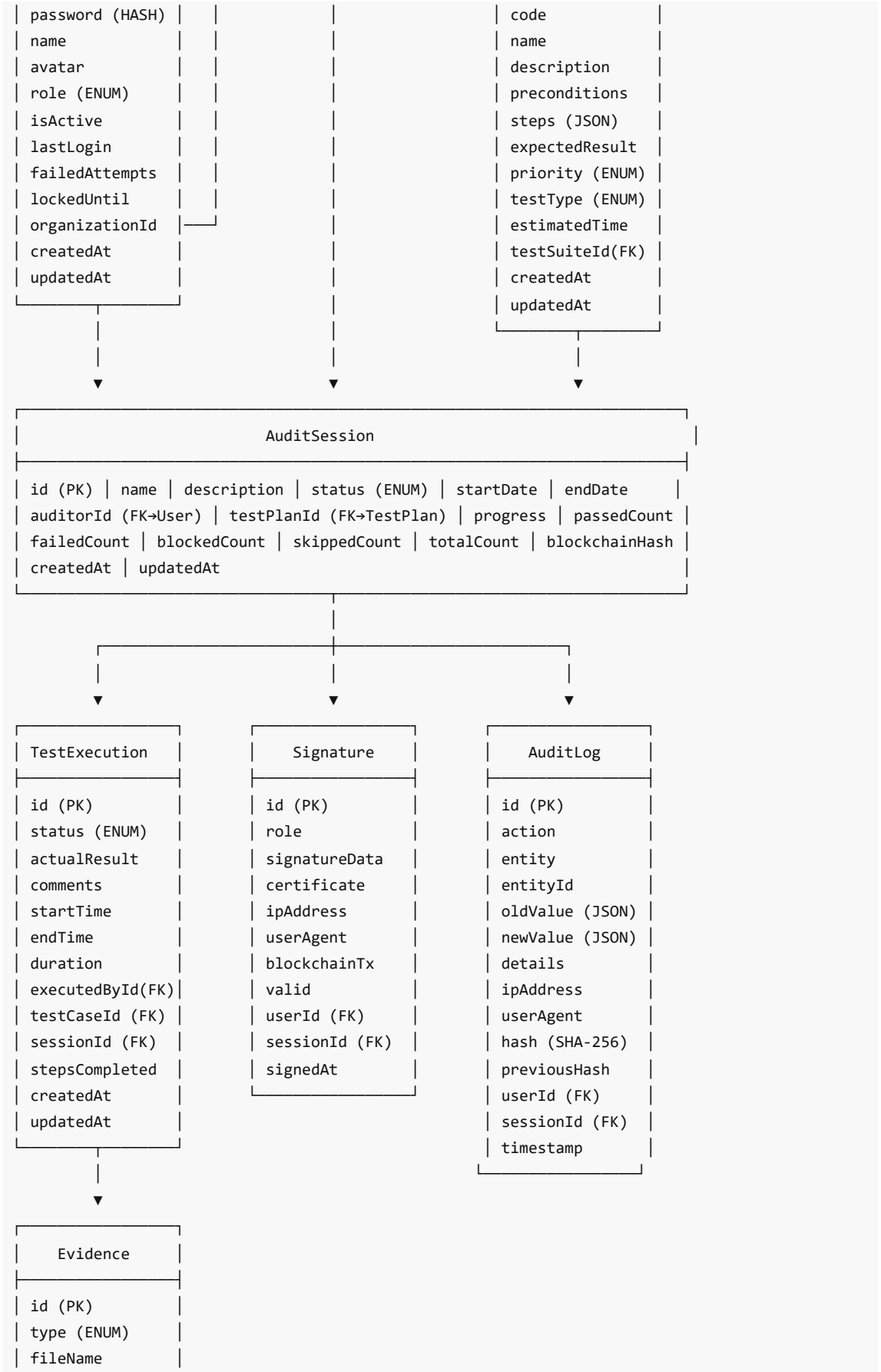
---

3. Modelo de Datos Relacional

3.1 Diagrama Entidad-Relacion (ERD)

El modelo de datos implementa un diseno normalizado hasta la Tercera Forma Normal (3NF) con desnormalizaciones estrategicas para optimizacion de consultas frecuentes.




password (HASH)
name
avatar
role (ENUM)
isActive
lastLogin
failedAttempts
lockedUntil
organizationId
createdAt
updatedAt
code
name
description
preconditions
steps (JSON)
expectedResult
priority (ENUM)
testType (ENUM)
estimatedTime
testSuiteId(FK)
createdAt
updatedAt
AuditSession
id (PK) | name | description | status (ENUM) | startDate | endDate
auditorId (FK→User) | testPlanId (FK→TestPlan) | progress | passedCount
failedCount | blockedCount | skippedCount | totalCount | blockchainHash
createdAt | updatedAt
TestExecution
id (PK)
status (ENUM)
actualResult
comments
startTime
endTime
duration
executedById(FK)
testCaseId (FK)
sessionId (FK)
stepsCompleted
createdAt
updatedAt
Signature
id (PK)
role
signatureData
certificate
ipAddress
userAgent
blockchainTx
valid
userId (FK)
sessionId (FK)
signedAt
AuditLog
id (PK)
action
entity
entityId
oldValue (JSON)
newValue (JSON)
details
ipAddress
userAgent
hash (SHA-256)
previousHash
userId (FK)
sessionId (FK)
timestamp
Evidence
id (PK)
type (ENUM)
fileName

```
| filePath        |
| fileSize        |
| mimeType        |
| hash (SHA-256)  |
| description     |
| metadata (JSON) |
| blockchainTx    |
| executionId(FK) |
| uploadedAt      |
└─────────────────┘
```

3.2 Enumeraciones del Dominio

Las enumeraciones representan el vocabulario controlado del dominio de auditoria de software:

```
// Roles de usuario siguiendo principio de minimo privilegio
enum UserRole {
  ADMIN        // Control total del sistema
  LEAD_AUDITOR // Crea sesiones, ejecuta pruebas, firma
  AUDITOR      // Ejecuta pruebas, agrega evidencias
  REVIEWER     // Solo lectura + firma de aprobacion
  VIEWER       // Solo lectura
}

// Estados del ciclo de vida de una sesion de auditoria
enum SessionStatus {
  DRAFT        // Borrador, no iniciada
  IN_PROGRESS  // En ejecucion activa
  REVIEW       // Pendiente de revision/firmas
  APPROVED     // Aprobada con todas las firmas
  REJECTED     // Rechazada, requiere re-trabajo
  ARCHIVED     // Archivada historicamente
}

// Estados de ejecucion de casos de prueba (IEEE 829)
enum ExecutionStatus {
  NOT_STARTED  // No iniciada
  IN_PROGRESS  // En ejecucion
  PASSED       // Aprobada - comportamiento esperado
  FAILED       // Fallida - defecto detectado
  BLOCKED      // Bloqueada - dependencia no satisfecha
  SKIPPED      // Omitida - fuera de alcance
}

// Prioridades de casos de prueba (ISTQB)
enum Priority {
  CRITICAL     // Funcionalidad critica del negocio
  HIGH         // Alta importancia, ejecucion obligatoria
  MEDIUM       // Importancia media
  LOW          // Baja prioridad, ejecucion opcional
}
```

```
// Tipos de prueba (ISO/IEC/IEEE 29119-4)
enum TestType {
  FUNCTIONAL   // Pruebas funcionales
  INTEGRATION  // Pruebas de integracion
  PERFORMANCE  // Pruebas de rendimiento
  SECURITY     // Pruebas de seguridad
  USABILITY    // Pruebas de usabilidad
  REGRESSION   // Pruebas de regresion
}

// Tipos de evidencia soportados
enum EvidenceType {
  SCREENSHOT   // Capturas de pantalla
  LOG          // Archivos de log
  PDF          // Documentos PDF
  VIDEO        // Grabaciones de video
  JSON         // Datos estructurados
  OTHER        // Otros tipos
}
```

3.3 Indices y Optimizaciones

El esquema define indices estrategicos para las consultas mas frecuentes:

```
model AuditLog {
  // ... campos ...

  @@index([userId])           // Consultas por usuario
  @@index([sessionId])        // Consultas por sesion
  @@index([entity, entityId]) // Consultas por entidad afectada
  @@index([timestamp])        // Consultas por rango temporal
}
```

---

4. Sistema de Autenticacion y Autorizacion

4.1 Arquitectura de Autenticacion

El sistema implementa autenticacion basada en **NextAuth.js v5** con estrategia de sesion **JWT (JSON Web Token)**, seleccionada por las siguientes razones tecnicas:

1. **Stateless**: No requiere almacenamiento de sesion en servidor
2. **Escalabilidad**: Permite escalamiento horizontal sin sesiones compartidas
3. **Compatibilidad Edge**: JWT funciona en Edge Runtime donde Prisma no esta disponible

```
// src/lib/auth.ts - Configuracion NextAuth
export const { handlers, signIn, signOut, auth } = NextAuth({
  adapter: PrismaAdapter(prisma) as any,
  session: {
    strategy: 'jwt',
    maxAge: 8 * 60 * 60, // 8 horas (jornada laboral)
```

```
},
pages: {
  signIn: '/login',
  error: '/login',
},
trustHost: true,
providers: [
  CredentialsProvider({
    name: 'credentials',
    credentials: {
      email: { label: 'Email', type: 'email' },
      password: { label: 'Password', type: 'password' },
    },
    async authorize(credentials) {
      // Validacion de entrada
      if (!credentials?.email || !credentials?.password) {
        throw new Error('Email y contrasena son requeridos')
      }

      // Busqueda de usuario
      const user = await prisma.user.findUnique({
        where: { email },
        include: { organization: true },
      })

      if (!user) {
        throw new Error('Credenciales invalidas')
      }

      // Verificacion de cuenta activa
      if (!user.isActive) {
        throw new Error('La cuenta esta desactivada')
      }

      // Verificacion de bloqueo temporal
      if (user.lockedUntil && user.lockedUntil > new Date()) {
        const minutesLeft = Math.ceil(
          (user.lockedUntil.getTime() - Date.now()) / 60000
        )
        throw new Error(
          `Cuenta bloqueada. Intente en ${minutesLeft} minutos`
        )
      }

      // Verificacion de contrasena con bcrypt
      const isPasswordValid = await bcrypt.compare(password, user.password)

      if (!isPasswordValid) {
        // Incremento de intentos fallidos con bloqueo progresivo
        const failedAttempts = user.failedAttempts + 1
        const updateData: { failedAttempts: number; lockedUntil?: Date } =
          { failedAttempts }
```

```
        // Bloqueo despues de 5 intentos fallidos
        if (failedAttempts >= 5) {
          updateData.lockedUntil = new Date(Date.now() + 15 * 60 * 1000)
          updateData.failedAttempts = 0
        }

        await prisma.user.update({
          where: { id: user.id },
          data: updateData,
        })

        throw new Error('Credenciales invalidas')
      }

      // Login exitoso: reset intentos y registro
      await prisma.user.update({
        where: { id: user.id },
        data: {
          failedAttempts: 0,
          lockedUntil: null,
          lastLogin: new Date(),
        },
      })

      // Registro en audit log
      await prisma.auditLog.create({
        data: {
          action: 'LOGIN',
          entity: 'User',
          entityId: user.id,
          userId: user.id,
          details: `Usuario ${user.email} inicio sesion exitosamente`,
        },
      })

      return {
        id: user.id,
        email: user.email,
        name: user.name,
        role: user.role,
        organizationId: user.organizationId,
        avatar: user.avatar,
      }
    },
  }),
],
callbacks: {
  async jwt({ token, user }) {
    if (user) {
      token.id = user.id as string
      token.role = user.role as string
```

```
        token.organizationId = user.organizationId as string
      }
      return token
    },
    async session({ session, token }) {
      if (token && session.user) {
        session.user.id = token.id as string
        session.user.role = token.role as string
        session.user.organizationId = token.organizationId as string
      }
      return session
    },
  },
})
```

4.2 Middleware de Proteccion de Rutas

El middleware implementa proteccion a nivel de Edge Runtime usando `getToken` de `next-auth/jwt`:

```
// src/middleware.ts
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'
import { getToken } from 'next-auth/jwt'

// Definicion de rutas publicas
const publicPaths = ['/', '/login', '/api/auth', '/demo', '/docs', '/support']

// Matriz de permisos RBAC
const roleProtectedPaths: Record<string, string[]> = {
  '/admin': ['ADMIN'],
  '/settings': ['ADMIN', 'LEAD_AUDITOR'],
}

export async function middleware(request: NextRequest) {
  const { pathname } = request.nextUrl

  // Permitir rutas publicas
  if (publicPaths.some(path => {
    if (path === '/') return pathname === '/'
    return pathname.startsWith(path)
  })) {
    return NextResponse.next()
  }

  // Permitir recursos estaticos
  if (
    pathname.startsWith('/_next') ||
    pathname.startsWith('/favicon') ||
    pathname.includes('.')
  ) {
    return NextResponse.next()
```

```
  }

  // Verificacion JWT (Edge Runtime compatible)
  const token = await getToken({
    req: request,
    secret: process.env.AUTH_SECRET
  })

  if (!token) {
    const loginUrl = new URL('/login', request.url)
    loginUrl.searchParams.set('callbackUrl', pathname)
    return NextResponse.redirect(loginUrl)
  }

  // Verificacion RBAC
  for (const [path, allowedRoles] of Object.entries(roleProtectedPaths)) {
    if (pathname.startsWith(path)) {
      if (!allowedRoles.includes(token.role as string)) {
        return NextResponse.redirect(new URL('/unauthorized', request.url))
      }
    }
  }

  return NextResponse.next()
}

export const config = {
  matcher: [
    '/((?!_next/static|_next/image|favicon.ico|api/auth).*)',
  ],
}
```

4.3 Matriz de Control de Acceso (RBAC)

Recurso / Accion	ADMIN	LEAD_AUDITOR	AUDITOR	REVIEWER	VIEWER
Ver Dashboard	✓	✓	✓	✓	✓
Crear Sesion	✓	✓	✗	✗	✗
Ejecutar Pruebas	✓	✓	✓	✗	✗
Agregar Evidencias	✓	✓	✓	✗	✗
Firmar Sesion	✓	✓	✗	✓	✗
Ver Audit Log	✓	✓	✗	✓	✗
Exportar Reportes	✓	✓	✓	✓	✓
Gestionar Usuarios	✓	✗	✗	✗	✗
Configuracion Sistema	✓	✓	✗	✗	✗

Eliminar Sesiones	✓	Solo DRAFT	✗	✗	✗

---

5. Mecanismos de Integridad Criptografica

5.1 Sistema de Audit Log con Encadenamiento Hash (Blockchain-like)

El sistema de audit log implementa un mecanismo de integridad basado en encadenamiento de hashes SHA-256, inspirado en la tecnologia blockchain pero sin el overhead de consenso distribuido:

```
// src/lib/audit-log.ts

/**
 * Crea un hash SHA-256 para un registro de audit log
 * El hash incluye los datos del registro Y el hash del registro anterior,
 * creando una cadena de integridad inmutable
 */
function createAuditHash(
  entry: AuditLogEntry,
  timestamp: Date,
  previousHash: string | null
): string {
  const data = JSON.stringify({
    action: entry.action,
    entity: entry.entity,
    entityId: entry.entityId,
    userId: entry.userId,
    sessionId: entry.sessionId || null,
    oldValue: entry.oldValue ? JSON.stringify(entry.oldValue) : null,
    newValue: entry.newValue ? JSON.stringify(entry.newValue) : null,
    changes: entry.changes ? JSON.stringify(entry.changes) : null,
    details: entry.details || null,
    timestamp: timestamp.toISOString(),
    previousHash: previousHash || 'GENESIS', // Bloque genesis
  })

  return createHash('sha256').update(data).digest('hex')
}

/**
 * Crea una entrada en el audit log con hash de integridad encadenado
 */
export async function createAuditLog(entry: AuditLogEntry) {
  // Obtener el ultimo registro para encadenar
  const lastLog = await prisma.auditLog.findFirst({
    orderBy: { timestamp: 'desc' },
    select: { hash: true },
  })

  const timestamp = new Date()
  const hash = createAuditHash(entry, timestamp, lastLog?.hash || null)
```

```
  return prisma.auditLog.create({
    data: {
      action: entry.action,
      entity: entry.entity,
      entityId: entry.entityId,
      userId: entry.userId,
      sessionId: entry.sessionId,
      oldValue: entry.oldValue ? JSON.stringify(entry.oldValue) : null,
      newValue: entry.newValue ? JSON.stringify(entry.newValue) : null,
      details: entry.details,
      ipAddress: entry.ipAddress || '127.0.0.1',
      userAgent: entry.userAgent,
      hash,                              // Hash actual
      previousHash: lastLog?.hash || null, // Hash anterior (encadenamiento)
      timestamp,
    },
  })
}
```

5.2 Verificacion de Integridad de la Cadena

```
/**
 * Verifica la integridad completa de la cadena de audit logs
 * Recalcula todos los hashes secuencialmente y compara
 */
export async function verifyAuditLogIntegrity(): Promise<{
  valid: boolean
  totalRecords: number
  invalidRecords: Array<{ id: string; expectedHash: string; actualHash: string }>
}> {
  const logs = await prisma.auditLog.findMany({
    orderBy: { timestamp: 'asc' },
  })

  const invalidRecords: Array<{
    id: string
    expectedHash: string
    actualHash: string
  }> = []

  let previousHash: string | null = null

  for (const log of logs) {
    const expectedHash = createAuditHash(
      {
        action: log.action,
        entity: log.entity,
        entityId: log.entityId,
        userId: log.userId,
        sessionId: log.sessionId || undefined,
```

```
        oldValue: log.oldValue ? JSON.parse(log.oldValue) : undefined,
        newValue: log.newValue ? JSON.parse(log.newValue) : undefined,
        details: log.details || undefined,
      },
      log.timestamp,
      previousHash
    )

    if (log.hash !== expectedHash) {
      invalidRecords.push({
        id: log.id,
        expectedHash,
        actualHash: log.hash || 'null',
      })
    }

    previousHash = log.hash
  }

  return {
    valid: invalidRecords.length === 0,
    totalRecords: logs.length,
    invalidRecords,
  }
}
```

5.3 Sistema de Firmas Digitales

Las firmas digitales garantizan no-repudio y autenticidad de las aprobaciones:

```
// src/lib/signature.ts

/**
 * Crea un hash unico del certificado de firma que incluye:
 * - Identidad del firmante
 * - Estado completo de la sesion al momento de firmar
 * - Timestamp exacto
 */
function createSignatureHash(data: {
  userId: string
  sessionId: string
  role: string
  timestamp: Date
  sessionData: string
}): string {
  const hashData = JSON.stringify({
    userId: data.userId,
    sessionId: data.sessionId,
    role: data.role,
    timestamp: data.timestamp.toISOString(),
    sessionData: data.sessionData,
```

```
  })

  return createHash('sha256').update(hashData).digest('hex')
}

/**
 * Crea una firma digital para una sesion de auditoria
 */
export async function createSignature(data: SignatureData) {
  // Obtener estado completo de la sesion para incluir en el hash
  const session = await prisma.auditSession.findUnique({
    where: { id: data.sessionId },
    include: {
      testPlan: { select: { title: true, version: true } },
      executions: {
        select: {
          status: true,
          testCase: { select: { code: true } },
        },
      },
    },
  })

  if (!session) {
    throw new Error('Sesion no encontrada')
  }

  // Serializar estado de la sesion para el hash
  const sessionData = JSON.stringify({
    name: session.name,
    status: session.status,
    testPlan: session.testPlan,
    passedCount: session.passedCount,
    failedCount: session.failedCount,
    blockedCount: session.blockedCount,
    skippedCount: session.skippedCount,
    totalCount: session.totalCount,
    progress: session.progress,
    executions: session.executions.map(e => ({
      status: e.status,
      testCode: e.testCase.code,
    })),
  })

  const timestamp = new Date()

  // Crear certificado (hash SHA-256)
  const certificate = createSignatureHash({
    userId: data.userId,
    sessionId: data.sessionId,
    role: data.role,
    timestamp,
```

```
    sessionData,
  })

  // Persistir firma
  const signature = await prisma.signature.create({
    data: {
      role: data.role,
      signatureData: data.signatureImage || null, // Imagen visual opcional
      certificate,                                // Hash criptografico
      ipAddress: data.ipAddress || '127.0.0.1',
      userAgent: data.userAgent,
      userId: data.userId,
      sessionId: data.sessionId,
      signedAt: timestamp,
    },
    include: {
      user: {
        select: { id: true, name: true, email: true, role: true },
      },
    },
  })

  // Registro en audit log
  await createAuditLog({
    action: AUDIT_ACTIONS.SIGNATURE_ADDED,
    entity: 'Signature',
    entityId: signature.id,
    userId: data.userId,
    sessionId: data.sessionId,
    details: `Firma agregada por ${user.name} como ${data.role}`,
    newValue: {
      role: data.role,
      certificate: certificate.substring(0, 16) + '...',
    },
  })

  return signature
}
```

5.4 Verificacion de Firma

```
/**
 * Verifica la integridad de una firma comparando el certificado
 * almacenado con un hash recalculado del estado actual
 *
 * IMPORTANTE: Si la sesion fue modificada despues de firmar,
 * la verificacion fallara, detectando alteracion
 */
export async function verifySignature(signatureId: string): Promise<{
  valid: boolean
  signature: { id: string; role: string; signedAt: Date; user: { name: string; email: string
```

```
} } | null
  reason?: string
}> {
  const signature = await prisma.signature.findUnique({
    where: { id: signatureId },
    include: {
      user: { select: { id: true, name: true, email: true } },
      session: {
        include: {
          testPlan: { select: { title: true, version: true } },
          executions: {
            select: {
              status: true,
              testCase: { select: { code: true } },
            },
          },
        },
      },
    },
  })

  if (!signature) {
    return { valid: false, signature: null, reason: 'Firma no encontrada' }
  }

  // Recalcular hash con estado actual
  const sessionData = JSON.stringify({
    name: signature.session.name,
    status: signature.session.status,
    testPlan: signature.session.testPlan,
    passedCount: signature.session.passedCount,
    failedCount: signature.session.failedCount,
    blockedCount: signature.session.blockedCount,
    skippedCount: signature.session.skippedCount,
    totalCount: signature.session.totalCount,
    progress: signature.session.progress,
    executions: signature.session.executions.map(e => ({
      status: e.status,
      testCode: e.testCase.code,
    })),
  })

  const expectedHash = createSignatureHash({
    userId: signature.userId,
    sessionId: signature.sessionId,
    role: signature.role,
    timestamp: signature.signedAt,
    sessionData,
  })

  const isValid = signature.certificate === expectedHash
```

```
  // Registrar verificacion
  await createAuditLog({
    action: AUDIT_ACTIONS.SIGNATURE_VERIFIED,
    entity: 'Signature',
    entityId: signatureId,
    userId: signature.userId,
    sessionId: signature.sessionId,
    details: `Verificacion de firma: ${isValid ? 'VALIDA' : 'INVALIDA - Posible
alteracion'}`,
  })

  return {
    valid: isValid,
    signature: {
      id: signature.id,
      role: signature.role,
      signedAt: signature.signedAt,
      user: signature.user,
    },
    reason: isValid ? undefined : 'Hash de certificado no coincide - datos alterados post-
firma',
  }
}
```

---

6. API RESTful y Endpoints

6.1 Catalogo de Endpoints

El sistema expone 17 endpoints organizados por dominio funcional:

Metodo	Endpoint	Descripcion	Roles Permitidos
Autenticacion			
POST	`/api/auth/[...nextauth]`	Endpoints NextAuth	Publico
Sesiones			
GET	`/api/sessions`	Listar sesiones con filtros	Todos autenticados
POST	`/api/sessions`	Crear nueva sesion	ADMIN, LEAD_AUDITOR
GET	`/api/sessions/[id]`	Obtener sesion por ID	Todos autenticados
PATCH	`/api/sessions/[id]`	Actualizar sesion	ADMIN, LEAD_AUDITOR, AUDITOR
DELETE	`/api/sessions/[id]`	Eliminar sesion (solo DRAFT)	ADMIN, LEAD_AUDITOR
Ejecuciones			

POST	`/api/executions`	Crear/actualizar ejecucion	ADMIN, LEAD_AUDITOR, AUDITOR
Evidencias			
GET	`/api/evidence`	Listar evidencias	Todos autenticados
POST	`/api/evidence`	Subir evidencia	ADMIN, LEAD_AUDITOR, AUDITOR
GET	`/api/evidence/[id]`	Obtener evidencia	Todos autenticados
DELETE	`/api/evidence/[id]`	Eliminar evidencia	ADMIN, LEAD_AUDITOR
Firmas			
GET	`/api/signatures`	Obtener firmas de sesion	ADMIN, LEAD_AUDITOR, REVIEWER
POST	`/api/signatures`	Agregar firma	ADMIN, LEAD_AUDITOR, REVIEWER
POST	`/api/signatures/[id]/verify`	Verificar firma	ADMIN, LEAD_AUDITOR, REVIEWER
Audit Log			
GET	`/api/audit-logs`	Consultar logs	ADMIN, LEAD_AUDITOR, REVIEWER
POST	`/api/audit-logs/verify`	Verificar integridad	ADMIN
Reportes			
GET	`/api/reports`	Listar reportes	Todos autenticados
POST	`/api/reports/generate`	Generar reporte	Todos autenticados
Dashboard			
GET	`/api/dashboard`	Estadisticas globales	Todos autenticados
Exportacion			
GET	`/api/export/session/[id]`	Exportar sesion completa	Todos autenticados

6.2 Ejemplo de Implementacion: GET /api/sessions/[id]

```
// src/app/api/sessions/[id]/route.ts

import { NextRequest, NextResponse } from 'next/server'
import { prisma } from '@/lib/prisma'

// GET /api/sessions/[id] - Obtener sesion por ID con estadisticas calculadas
export async function GET(
  request: NextRequest,
```

```
  { params }: { params: Promise<{ id: string }> }
) {
  try {
    const { id } = await params

    const session = await prisma.auditSession.findUnique({
      where: { id },
      include: {
        auditor: true,
        testPlan: {
          include: {
            testSuites: {
              include: {
                testCases: true,
              },
              orderBy: {
                order: 'asc',
              },
            },
          },
        },
        executions: {
          include: {
            testCase: true,
            executedBy: true,
            evidences: true,
          },
        },
        signatures: {
          include: {
            user: true,
          },
        },
      },
    })

    if (!session) {
      return NextResponse.json(
        { error: 'Sesion no encontrada' },
        { status: 404 }
      )
    }

    // IMPORTANTE: Calcular estadisticas en tiempo real
    // No confiar en valores almacenados que pueden estar desactualizados
    const passedCount = session.executions.filter(e => e.status === 'PASSED').length
    const failedCount = session.executions.filter(e => e.status === 'FAILED').length
    const blockedCount = session.executions.filter(e => e.status === 'BLOCKED').length
    const skippedCount = session.executions.filter(e => e.status === 'SKIPPED').length
    const totalCount = session.testPlan?.testSuites.reduce(
      (acc, suite) => acc + suite.testCases.length, 0
    ) || 0
```

```
    const executedCount = passedCount + failedCount + blockedCount + skippedCount
    const progress = totalCount > 0 ? Math.round((executedCount / totalCount) * 100) : 0

    return NextResponse.json({
      ...session,
      passedCount,
      failedCount,
      blockedCount,
      skippedCount,
      totalCount,
      progress,
    })
  } catch (error) {
    console.error('Error fetching session:', error)
    return NextResponse.json(
      { error: 'Error al obtener la sesion' },
      { status: 500 }
    )
  }
}
```

---

7. Componentes de Interfaz de Usuario

7.1 Arquitectura de Componentes

El sistema utiliza **React Server Components (RSC)** como paradigma principal, con Client Components para interactividad:

7.2 Biblioteca de Componentes UI

El sistema utiliza **Shadcn/UI** como biblioteca de componentes, basada en:

- **Radix UI**: Primitivos accesibles (ARIA compliant)
- **Tailwind CSS**: Utility-first styling
- **CVA (Class Variance Authority)**: Variantes de componentes

Componentes principales utilizados:

Componente	Uso	Archivo
`Card`	Contenedores de informacion	Dashboard, Sessions
`Button`	Acciones primarias/secundarias	Formularios, navegacion
`Input`	Campos de texto	Login, filtros
`Badge`	Estados y etiquetas	Status de sesiones
`Progress`	Barras de progreso	Ejecucion de pruebas
`Table`	Tablas de datos	Audit log, sesiones
`Dialog`	Modales	Confirmaciones, firmas

7.3 Graficos y Visualizaciones

El dashboard implementa visualizaciones con **Recharts**:

```
// src/components/dashboard/charts.tsx

// Grafico de torta para estados de ejecucion
export function ExecutionStatusPieChart({ data }: { data: ExecutionStats }) {
  const chartData = [
    { name: 'Aprobadas', value: data.passed, fill: '#10b981' },
    { name: 'Fallidas', value: data.failed, fill: '#ef4444' },
    { name: 'Bloqueadas', value: data.blocked, fill: '#f59e0b' },
    { name: 'Omitidas', value: data.skipped, fill: '#6b7280' },
    { name: 'Pendientes', value: data.pending, fill: '#e5e7eb' },
  ]

  return (
    <ResponsiveContainer width="100%" height={200}>
      <PieChart>
        <Pie
          data={chartData}
          dataKey="value"
          nameKey="name"
          cx="50%"
          cy="50%"
          outerRadius={80}
          label
        />
```

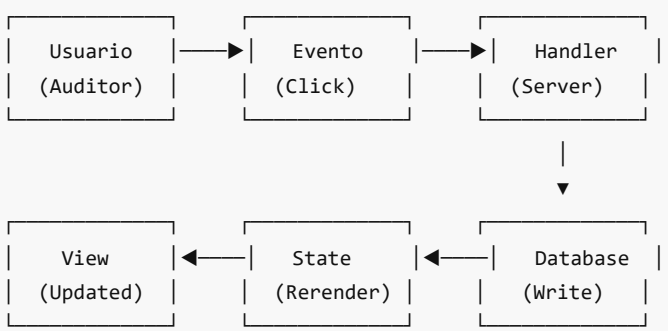
```
        <Tooltip />
        <Legend />
      </PieChart>
    </ResponsiveContainer>
  )
}

// Grafico de linea para tendencia temporal
export function ExecutionTrendChart({ data }: { data: TrendData[] }) {
  return (
    <ResponsiveContainer width="100%" height={200}>
      <LineChart data={data}>
        <CartesianGrid strokeDasharray="3 3" />
        <XAxis dataKey="date" />
        <YAxis />
        <Tooltip />
        <Line type="monotone" dataKey="passed" stroke="#10b981" name="Aprobadas" />
        <Line type="monotone" dataKey="failed" stroke="#ef4444" name="Fallidas" />
      </LineChart>
    </ResponsiveContainer>
  )
}
```

---

8. Flujo de Datos y Patrones de Diseno

8.1 Patron de Flujo Unidireccional de Datos



8.2 Patron Repository (Prisma como Implementation)

```
// Ejemplo de patron Repository implementado con Prisma

// Repository Interface (implicito en TypeScript)
interface SessionRepository {
  findById(id: string): Promise<Session | null>
  findAll(filters: SessionFilters): Promise<Session[]>
  create(data: CreateSessionDTO): Promise<Session>
  update(id: string, data: UpdateSessionDTO): Promise<Session>
  delete(id: string): Promise<void>
```

```
}

// Implementacion con Prisma (directo en route handlers)
const session = await prisma.auditSession.findUnique({
  where: { id },
  include: {
    auditor: true,
    testPlan: { include: { testSuites: { include: { testCases: true } } } },
    executions: { include: { testCase: true, evidences: true } },
    signatures: { include: { user: true } },
  },
})
```

8.3 Patron Observer para Audit Log

Cada operacion de escritura dispara automaticamente la creacion de un registro de auditoria:

```
// Ejemplo: Creacion de firma con registro automatico

const signature = await prisma.signature.create({ data: signatureData })

// Observer: Registro automatico en audit log
await createAuditLog({
  action: AUDIT_ACTIONS.SIGNATURE_ADDED,
  entity: 'Signature',
  entityId: signature.id,
  userId: data.userId,
  sessionId: data.sessionId,
  details: `Firma agregada por ${user.name} como ${data.role}`,
  newValue: { role: data.role, certificate: certificate.substring(0, 16) + '...' },
})
```

---

9. Seguridad y Cumplimiento

9.1 OWASP Top 10 Mitigaciones

Vulnerabilidad	Mitigacion Implementada
A01:2021 Broken Access Control	RBAC en middleware + verificacion en cada endpoint
A02:2021 Cryptographic Failures	bcrypt para passwords, SHA-256 para hashes
A03:2021 Injection	Prisma ORM con queries parametrizadas
A04:2021 Insecure Design	Arquitectura en capas con separacion de concerns
A05:2021 Security Misconfiguration	Variables de entorno para secretos
A06:2021 Vulnerable Components	Dependencias actualizadas (npm audit)
A07:2021 Authentication Failures	Bloqueo progresivo, sesiones JWT temporales

A08:2021 Software Integrity	Hash chain en audit log, firmas verificables
A09:2021 Security Logging	Audit log completo con hash de integridad
A10:2021 SSRF	No se permite input de URLs externos

9.2 Proteccion contra Ataques de Fuerza Bruta

```
// Implementacion de rate limiting y bloqueo temporal

// En authorize():
if (user.lockedUntil && user.lockedUntil > new Date()) {
  const minutesLeft = Math.ceil(
    (user.lockedUntil.getTime() - Date.now()) / 60000
  )
  throw new Error(`Cuenta bloqueada. Intente en ${minutesLeft} minutos`)
}

// Incremento progresivo de intentos fallidos
const failedAttempts = user.failedAttempts + 1
if (failedAttempts >= 5) {
  updateData.lockedUntil = new Date(Date.now() + 15 * 60 * 1000) // 15 min
  updateData.failedAttempts = 0
}
```

9.3 Cumplimiento ISO/IEC/IEEE 29119

El sistema cumple con la estructura de documentacion de pruebas definida en ISO/IEC/IEEE 29119-3:

Artefacto 29119	Implementacion AT5
Test Plan	Modelo `TestPlan` con scope, objectives, strategy
Test Suite	Modelo `TestSuite` con categoria y ordenamiento
Test Case	Modelo `TestCase` con steps, expected result, priority
Test Log	Modelo `TestExecution` con timestamps y duracion
Test Report	Modelo `Report` con multiples formatos
Incident Report	Ejecuciones con status FAILED + evidencias

---

10. Consideraciones de Rendimiento

10.1 Optimizaciones Implementadas

1. **Server Components por defecto**: Reduccion de bundle JavaScript en cliente
2. **Streaming con Suspense**: Renderizado progresivo de componentes
3. **Conexiones Prisma pooled**: Reutilizacion de conexiones a base de datos
4. **Indices en AuditLog**: Optimizacion de queries frecuentes
5. **Calculo de estadisticas en servidor**: Evita multiples round-trips

10.2 Metricas de Rendimiento Esperadas

Operacion	Tiempo Objetivo	Observaciones
Login	< 500ms	Incluye bcrypt.compare (10 rounds)
Dashboard load	< 1s	7 queries paralelas con Promise.all
Session detail	< 500ms	Include nested con 4 niveles
Create execution	< 200ms	Escritura simple + audit log
Verify signature	< 300ms	Recalculo de hash SHA-256

---

11. Referencias Bibliograficas

1. Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley.
2. IEEE Computer Society. (2008). *IEEE Standard for Software Reviews and Audits* (IEEE 1028-2008).
3. ISO/IEC/IEEE. (2013). *Software and systems engineering - Software testing - Part 3: Test documentation* (ISO/IEC/IEEE 29119-3:2013).
4. ISTQB. (2023). *Certified Tester Foundation Level Syllabus v4.0*. International Software Testing Qualifications Board.
5. OWASP Foundation. (2021). *OWASP Top 10:2021 - The Ten Most Critical Web Application Security Risks*.
6. Prisma. (2024). *Prisma Documentation*. https://www.prisma.io/docs
7. Vercel. (2024). *Next.js 15 Documentation*. https://nextjs.org/docs
8. Auth.js. (2024). *NextAuth.js v5 Documentation*. https://authjs.dev
9. Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Conceptos de blockchain aplicados a audit log]
10. Martin, R.C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.

Apendice A: Glosario de Terminos

Termino	Definicion
Audit Log	Registro cronologico inmutable de todas las acciones del sistema
RBAC	Role-Based Access Control - Control de acceso basado en roles
JWT	JSON Web Token - Token de autenticacion stateless
SHA-256	Secure Hash Algorithm - Funcion hash criptografica de 256 bits
ORM	Object-Relational Mapping - Capa de abstraccion de base de datos
RSC	React Server Components - Componentes renderizados en servidor

V&V	Verification and Validation - Procesos de aseguramiento de calidad

---

Apendice B: Acciones de Auditoria Estandarizadas

```typescript
export const AUDIT_ACTIONS = {
  // Autenticacion
  LOGIN: 'LOGIN',
  LOGOUT: 'LOGOUT',
  LOGIN_FAILED: 'LOGIN_FAILED',
  PASSWORD_CHANGED: 'PASSWORD_CHANGED',

  // Sesiones de auditoria
  SESSION_CREATED: 'SESSION_CREATED',
  SESSION_UPDATED: 'SESSION_UPDATED',
  SESSION_DELETED: 'SESSION_DELETED',
  SESSION_STARTED: 'SESSION_STARTED',
  SESSION_COMPLETED: 'SESSION_COMPLETED',
  SESSION_APPROVED: 'SESSION_APPROVED',
  SESSION_REJECTED: 'SESSION_REJECTED',

  // Ejecucion de pruebas
  TEST_EXECUTED: 'TEST_EXECUTED',
  TEST_PASSED: 'TEST_PASSED',
  TEST_FAILED: 'TEST_FAILED',
  TEST_BLOCKED: 'TEST_BLOCKED',
  TEST_SKIPPED: 'TEST_SKIPPED',
  EXECUTION_UPDATED: 'EXECUTION_UPDATED',

  // Evidencias
  EVIDENCE_UPLOADED: 'EVIDENCE_UPLOADED',
  EVIDENCE_DELETED: 'EVIDENCE_DELETED',

  // Firmas
  SIGNATURE_ADDED: 'SIGNATURE_ADDED',
  SIGNATURE_VERIFIED: 'SIGNATURE_VERIFIED',

  // Reportes
  REPORT_GENERATED: 'REPORT_GENERATED',
  REPORT_EXPORTED: 'REPORT_EXPORTED',
  REPORT_DOWNLOADED: 'REPORT_DOWNLOADED',

  // Administracion
  USER_CREATED: 'USER_CREATED',
  USER_UPDATED: 'USER_UPDATED',
  USER_DEACTIVATED: 'USER_DEACTIVATED',
  SETTINGS_CHANGED: 'SETTINGS_CHANGED',
} as const
```

Fin del Documento 1: Arquitectura Tecnica