


Sistema de Versionado - EscalaFin MVP








Versión actual: 1.1.0
Build: 20251030.001
Fecha: 30 de Octubre de 2025
Estado:  IMPLEMENTADO Y ACTIVO

Índice

- 1. [Resumen](#)
- 2. [Archivos del Sistema](#)
- 3. [Semantic Versioning](#)
- 4. [Cómo Actualizar la Versión](#)
- 5. [API de Versión](#)
- 6. [Componente UI](#)
- 7. [Integración con CI/CD](#)
- 8. [Ejemplos de Uso](#)

Resumen

Se ha implementado un sistema completo de versionado para EscalaFin MVP que permite:

-  **Identificar versiones deployadas** con precisión
-  **Tracking de cambios** mediante CHANGELOG
-  **Consulta programática** de versión vía API
-  **Visualización en UI** con componente React
-  **Actualización automatizada** con script bash
-  **Git tags** para releases importantes
-  **Build numbers** únicos por deploy

Archivos del Sistema

1. VERSION

Ubicación: `/VERSION` (raíz del proyecto)

Archivo simple con el número de versión:

```
1.1.0
```

Uso: Lectura rápida de versión en scripts y despliegues.

2. version.json

Ubicaciones:

- /version.json (raíz del proyecto)
- /app/version.json (dentro de la app)

Metadata detallada de la versión:

```
{
  "version": "1.1.0",
  "buildNumber": "20251030.001",
  "releaseDate": "2025-10-30",
  "releaseName": "Portability & Deploy Fixes",
  "gitCommit": "72e5437",
  "environment": "production",
  "changelog": [
    "Eliminadas todas las rutas absolutas hardcodeadas",
    "Corregida configuración de Prisma para portabilidad",
    "..."
  ],
  "compatibility": {
    "node": "18.x",
    "npm": "9.x",
    "nextjs": "14.2.28",
    "prisma": "6.7.0",
    "docker": ">=20.10"
  },
  "deployments": {
    "docker": "compatible",
    "easypanel": "compatible",
    "coolify": "compatible",
    "kubernetes": "compatible"
  }
}
```

Uso: Información completa para deployment y debugging.

3. CHANGELOG.md

Ubicación: /CHANGELOG.md

Historial completo de cambios siguiendo [Keep a Changelog](https://keepachangelog.com/) (<https://keepachangelog.com/>):

```
# Changelog

## [1.1.0] - 2025-10-30

### 🛠 Corregido (Fixed)
- Rutas absolutas eliminadas
- Prisma Schema corregido
- ...

## [1.0.0] - 2025-10-29

### 🎉 Lanzamiento Inicial
- Características principales
- ...
```

Uso: Documentación de cambios para usuarios y desarrolladores.

4. API Endpoint

Ubicación: `/app/app/api/system/version/route.ts`

Endpoint REST para consultar versión programáticamente:

Endpoint: `GET /api/system/version`

Respuesta:

```
{
  "version": "1.1.0",
  "buildNumber": "20251030.001",
  "releaseDate": "2025-10-30",
  "releaseName": "Portability & Deploy Fixes",
  "gitCommit": "72e5437",
  "environment": "production",
  "timestamp": "2025-10-30T00:56:00Z",
  "nodeVersion": "v18.17.0",
  "platform": "linux",
  "uptime": 12345.67
}
```

Uso: Monitoreo, debugging, health checks.

5. Componente UI

Ubicación: `/app/components/layout/version-info.tsx`

Componente React que muestra la versión en la interfaz:

```
import { VersionInfo } from '@components/layout/version-info'

// En tu layout:
<VersionInfo />
```

Características:

- Badge con número de versión
- Tooltip con información detallada
- Actualización automática desde API
- Indicador de entorno (producción/desarrollo)

Uso: Footer, sidebar, o página de configuración.

6. Script de Actualización

Ubicación: `/scripts/update-version.sh`

Script bash para actualizar versión automáticamente:

```
#!/bin/bash
./scripts/update-version.sh [major|minor|patch] "Descripción"
```

Funciones:

- Actualiza package.json
- Actualiza VERSION

- Actualiza version.json (ambos)
 - Agrega entrada a CHANGELOG.md
 - Opcionalmente crea commit y tag
 - Genera build number automáticamente
-

Semantic Versioning

El proyecto sigue [Semantic Versioning 2.0.0](https://semver.org/) (<https://semver.org/>):

Formato: MAJOR.MINOR.PATCH

MAJOR (1.0.0 → 2.0.0)

- Cambios incompatibles con versiones anteriores
- Breaking changes en API
- Cambios arquitecturales importantes

Ejemplos:

- Nueva versión de Next.js con breaking changes
- Reestructuración completa de base de datos
- Cambios incompatibles en API endpoints

MINOR (1.0.0 → 1.1.0)

- Nuevas funcionalidades compatibles
- Mejoras que no rompen código existente
- Nuevos módulos o características

Ejemplos:

- Nuevo módulo de reportes
- Integración con nuevo sistema de pagos
- Nuevas funcionalidades de dashboard

PATCH (1.0.0 → 1.0.1)

- Corrección de bugs
- Fixes de seguridad
- Mejoras de rendimiento
- Cambios de documentación

Ejemplos:

- Fix de bug en cálculo de intereses
- Corrección de ruta que causaba 404
- Mejora de rendimiento en queries

Build Numbers

Formato: **YYYYMMDD.NNN**

- **YYYYMMDD**: Fecha del build (20251030)
- **NNN**: Número secuencial del día (001, 002, 003, ...)

Ejemplos:

- 20251030.001 - Primer build del 30 de octubre
- 20251030.002 - Segundo build del mismo día
- 20251031.001 - Primer build del 31 de octubre

Uso:

- Identificación única de cada deploy
- Tracking en logs y monitoreo
- Rollback a build específico

Cómo Actualizar la Versión

Método 1: Script Automatizado (Recomendado)

```
cd /home/ubuntu/escalafin_mvp

# Para un PATCH (1.1.0 → 1.1.1)
./scripts/update-version.sh patch "Fix de bug en pagos"

# Para un MINOR (1.1.0 → 1.2.0)
./scripts/update-version.sh minor "Nuevo módulo de reportes avanzados"

# Para un MAJOR (1.1.0 → 2.0.0)
./scripts/update-version.sh major "Migración a Next.js 15"
```

El script hará:

1. ☒ Actualizar package.json
2. ☒ Actualizar VERSION
3. ☒ Actualizar version.json (ambos)
4. ☒ Agregar entrada a CHANGELOG.md
5. ☒ Generar build number automáticamente
6. ☒ Preguntar si crear commit
7. ☒ Preguntar si crear tag
8. ☒ Mostrar comandos para push

Ejemplo de salida:



Método 2: Manual

Si prefieres actualizar manualmente:

```
cd /home/ubuntu/escalafin_mvp

# 1. Actualizar package.json
cd app
npm version minor --no-git-tag-version # o patch, major
cd ..

# 2. Actualizar VERSION
echo "1.2.0" > VERSION

# 3. Actualizar version.json (editar manualmente)
# Actualizar ambos: /version.json y /app/version.json

# 4. Actualizar CHANGELOG.md (editar manualmente)

# 5. Commit y tag
git add -A
git commit -m "🚀 Release v1.2.0: Descripción"
git tag -a v1.2.0 -m "Release v1.2.0"

# 6. Push
git push origin main
git push origin v1.2.0
```

API de Versión

Endpoint

```
GET /api/system/version
```

Respuesta

```
{
  "version": "1.1.0",
  "buildNumber": "20251030.001",
  "releaseDate": "2025-10-30",
  "releaseName": "Portability & Deploy Fixes",
  "gitCommit": "72e5437",
  "environment": "production",
  "changelog": [
    "Eliminadas todas las rutas absolutas hardcodeadas",
    "Corregida configuración de Prisma para portabilidad",
    "Eliminado yarn.lock (proyecto usa NPM)",
    "..."
  ],
  "compatibility": {
    "node": "18.x",
    "npm": "9.x",
    "nextjs": "14.2.28",
    "prisma": "6.7.0",
    "docker": ">=20.10"
  },
  "deployments": {
    "docker": "compatible",
    "easypanel": "compatible",
    "coolify": "compatible",
    "kubernetes": "compatible"
  },
  "timestamp": "2025-10-30T00:56:00.000Z",
  "nodeVersion": "v18.17.0",
  "platform": "linux",
  "uptime": 12345.67
}
```

Uso desde JavaScript

```
// Consultar versión
const response = await fetch('/api/system/version')
const versionInfo = await response.json()
console.log(`Running version: ${versionInfo.version}`)
console.log(`Build: ${versionInfo.buildNumber}`)
console.log(`Commit: ${versionInfo.gitCommit}`)
```

Uso desde curl

```
curl https://tu-dominio.com/api/system/version | jq
```

Componente UI

Importación

```
import { VersionInfo } from '@components/layout/version-info'
```


Uso en Layout

```
// app/layout.tsx
export default function RootLayout({ children }) {
  return (
    <html>
      <body>
        <div className="app-container">
          {children}
        </div>

        <footer className="app-footer">
          <VersionInfo />
        </footer>
      </body>
    </html>
  )
}
```

Uso en Sidebar

```
// components/sidebar.tsx
export function Sidebar() {
  return (
    <aside className="sidebar">
      <nav>
        { /* Navigation items */ }
      </nav>

      <div className="sidebar-footer">
        <VersionInfo />
      </div>
    </aside>
  )
}
```

Personalización

El componente usa Tailwind CSS y puede personalizarse:

```
<div className="flex items-center gap-2">
  <VersionInfo />
  <span className="text-xs text-muted-foreground">
    © 2025 EscalaFin
  </span>
</div>
```

Integración con CI/CD

GitHub Actions

```
name: Deploy

on:
  push:
    tags:
      - 'v*'

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Get version
        id: version
        run: |
          VERSION=$(cat VERSION)
          echo "version=$VERSION" >> $GITHUB_OUTPUT

      - name: Build Docker image
        run: |
          docker build -t escalafin:${{ steps.version.outputs.version }} .

      - name: Deploy to production
        run: |
          echo "Deploying version ${ steps.version.outputs.version }"
          # Deploy commands here
```

EasyPanel/Coolify

El sistema detecta automáticamente la versión desde los archivos:

1. **VERSION** se copia al contenedor Docker
2. **version.json** está disponible en runtime
3. **API endpoint** responde con versión actual
4. **Health checks** pueden consultar `/api/system/version`

Docker

El Dockerfile incluye automáticamente los archivos de versión:

```
# Los archivos se copian con COPY . .
COPY . .

# Verificar versión en el build
RUN cat VERSION && \
    cat version.json | jq -r '.version'
```

Health Check

```
#!/bin/bash
# healthcheck.sh

# Verificar que la app responde
curl -f http://localhost:3000/api/system/version || exit 1

# Opcional: Verificar versión específica
VERSION=$(curl -s http://localhost:3000/api/system/version | jq -r '.version')
echo "Running version: $VERSION"
```

Ejemplos de Uso

Ejemplo 1: Release de Bug Fix

```
./scripts/update-version.sh patch "Fix de cálculo de intereses en préstamos"
# Version: 1.1.0 → 1.1.1

git push origin main
git push origin v1.1.1
```

Ejemplo 2: Release de Nueva Funcionalidad

```
./scripts/update-version.sh minor "Agregado módulo de reportes avanzados con gráficas"
# Version: 1.1.0 → 1.2.0

git push origin main
git push origin v1.2.0
```

Ejemplo 3: Release Mayor

```
./scripts/update-version.sh major "Migración a Next.js 15 y React 19"
# Version: 1.2.0 → 2.0.0

git push origin main
git push origin v2.0.0
```

Ejemplo 4: Consultar Versión en Logs

```
// En cualquier parte de tu código
import fs from 'fs'
import path from 'path'

function logVersionInfo() {
  const versionPath = path.join(process.cwd(), 'version.json')
  const version = JSON.parse(fs.readFileSync(versionPath, 'utf-8'))

  console.log('=' .repeat(60))
  console.log(`EscalaFin MVP v${version.version}`)
  console.log(`Build: ${version.buildNumber}`)
  console.log(`Commit: ${version.gitCommit}`)
  console.log(`Environment: ${version.environment}`)
  console.log('=' .repeat(60))
}

// Llamar al inicio de la app
logVersionInfo()
```

Ejemplo 5: Mostrar Versión en Error Pages





```
// app/error.tsx
import { VersionInfo } from '@components/layout/version-info'

export default function Error({ error }) {
  return (
    <div className="error-page">
      <h1>Oops! Algo salió mal</h1>
      <p>{error.message}</p>




      <div className="error-footer">
        <p>Por favor, incluye esta información al reportar el error:</p>
        <VersionInfo />
      </div>
    </div>
  )
}
```

Ventajas del Sistema

Para Desarrollo





-  Tracking preciso de cambios
-  Fácil identificación de qué código está en cada ambiente
-  Rollback simple a versión anterior
-  Documentación automática de cambios

Para Deploy





-  Identificación única de cada build
-  Verificación de versión deployada
-  Health checks informativos

-  Logs con contexto de versión

Para Debugging

-  Reproducción de bugs en versión específica
-  Comparación entre versiones
-  Información completa en reportes de error
-  Auditoría de cambios

Para Usuarios

-  Transparencia de versión en uso
-  Información en reportes de bugs
-  Conocimiento de nuevas características
-  Historial de cambios disponible



Referencias

- [Semantic Versioning](https://semver.org/) (https://semver.org/)
- [Keep a Changelog](https://keepachangelog.com/) (https://keepachangelog.com/)
- [Conventional Commits](https://www.conventionalcommits.org/) (https://www.conventionalcommits.org/)
- [Git Tags](https://git-scm.com/book/en/v2/Git-Basics-Tagging) (https://git-scm.com/book/en/v2/Git-Basics-Tagging)



Troubleshooting

El script de actualización no funciona

```
# Asegurar que el script es ejecutable
chmod +x scripts/update-version.sh

# Verificar que estás en la raíz del proyecto
cd /home/ubuntu/escalafin_mvp
```

La API no devuelve la versión

```
# Verificar que version.json existe
ls -la app/version.json

# Verificar contenido
cat app/version.json | jq
```

El componente UI no se muestra

```
// Verificar que está importado correctamente
import { VersionInfo } from '@components/layout/version-info'

// Verificar que el componente está en el layout
<VersionInfo />
```

Conflicto de versiones en package.json

```
# Resetear y actualizar  
cd app  
npm version 1.1.0 --no-git-tag-version --allow-same-version  
cd ..
```

Última actualización: 30 de Octubre de 2025

Versión del documento: 1.0

Commit: 72e5437