SECRETARÍA
DE INNOVACIÓN

Guía de desarrollo de software

Estándares y buenas prácticas

v. 2025.1

Creado por
Equipo I+D+i

12 de marzo de 2025

Contenido

Introducción

En el ámbito del desarrollo de software, la estandarización y las buenas prácticas constituyen pilares fundamentales para garantizar la calidad, mantenibilidad y escalabilidad de las aplicaciones. El presente documento establece los estándares y lineamientos oficiales que deberán implementar y seguir todos los equipos de desarrollo de la Secretaría de Innovación de la Presidencia.

La ausencia de estándares unificados en los procesos de desarrollo ha generado una serie de problemáticas significativas:

- Código de difícil mantenimiento y comprensión.
- Duplicación sistemática de funcionalidades entre proyectos.
- Incidencia elevada de errores en entornos productivos.
- Documentación técnica insuficiente o inexistente.
- Redundancia de código y soluciones.
- Dificultades para la transferencia de conocimiento.

Estas dificultades no solo afectan la calidad del producto final entregado a las instituciones y/o ciudadanos, sino que también impactan negativamente en la eficiencia operativa de los equipos, generando costos adicionales y retrasos en la implementación de nuevas funcionalidades prioritarias para la administración pública.

La implementación de los estándares contenidos en este documento busca mejorar significativamente la productividad de los equipos, reducir la incidencia de errores en producción y facilitar la colaboración efectiva entre desarrolladores, independientemente del proyecto en el que se desempeñen. Los beneficios esperados incluyen:

1. Optimización de recursos técnicos y humanos.
2. Reducción de tiempos de desarrollo y mantenimiento.
3. Mejora en la calidad general del software producido.
4. Facilitación de la incorporación de nuevos profesionales a los equipos.

5. Establecimiento de métricas objetivas de calidad.

Se reconoce que, hasta el presente, cada equipo ha utilizado plantillas diferentes o ha iniciado proyectos desde cero según criterios individuales, sin una directriz institucional clara. En respuesta a esta situación, se han desarrollado estándares y plantillas unificadas para las diferentes tecnologías utilizadas en la Secretaría. Esta iniciativa tiene como objetivo estandarizar el desarrollo, garantizando coherencia metodológica y aplicación de buenas prácticas en todos los proyectos, independientemente del lenguaje o framework empleado.

Los lineamientos contenidos en este documento han sido elaborados considerando las mejores prácticas de la industria, adaptadas a las necesidades y contexto específicos de la administración pública, y su cumplimiento será objeto de evaluación periódica como parte de los indicadores de desempeño técnico de los equipos.

Alcance

Establecer un conjunto de estándares y buenas prácticas para el desarrollo de software en la Secretaría de Innovación de la Presidencia, mediante la implementación de una plantilla desarrollada en Laravel que promueva la calidad, mantenibilidad y escalabilidad de las aplicaciones.

El presente documento se ha desarrollado contemplando el siguiente contenido:

1. La implementación de plantillas de desarrollo que incorporen estándares y buenas prácticas.
2. La estandarización de la estructura del código, facilitando la colaboración entre equipos.
3. La promoción de buenas prácticas, mejorando la calidad y mantenibilidad del software.
4. La implementación de pruebas automatizadas, reduciendo errores en producción.

Estándares

Este documento establece las buenas prácticas que deben seguirse en la programación, seguridad, mantenimiento y calidad del código fuente, con el objetivo de garantizar su legibilidad, eficiencia y escalabilidad.

Estos estándares son de aplicación obligatoria para todos los desarrolladores y equipos de tecnología involucrados en la creación del software institucional, su cumplimiento es fundamental para facilitar la colaboración, reducir errores y mejorar la seguridad de las aplicaciones desarrolladas.

A continuación, se presentan los estándares que deben implementarse en el desarrollo de software:

- Deben utilizarse nombres descriptivos para clases, métodos, objetos u otros elementos.
- Las variables deben ser nombradas de acuerdo con su función en el código fuente.
- Todo el código fuente debe estar claramente tabulado o formateado.
- Deben realizarse pruebas unitarias en cada uno de los módulos, verificando que estos cumplan con los requerimientos del software.
- Los comentarios del código deben ser concisos y breves a menos que sean métodos complejos.
- Deben comentarse los distintos bloques de código, aplicando un criterio uniforme para cada nivel y siguiendo un modelo basado en los siguientes aspectos:
    - Incluir en cada clase una breve descripción, autor y fecha de última modificación.
    - Incluir por cada método, una descripción de su objeto y funcionalidades, así como de los parámetros y resultados obtenidos.
- Los comentarios en el código fuente deben mantenerse actualizados:
    - Si en algún momento la funcionalidad del código cambia, deben actualizarse los comentarios.

    - De cambiar la naturaleza de algún proceso, debe actualizarse inmediatamente el comentario asociado.
- Debe mantenerse el mismo estilo de formato y comentarios en el código fuente para permitir una mejor comprensión.
- No deben utilizarse palabras o frases indebidas en los nombres de entidades, métodos o comentarios dentro del código fuente.
- No deben almacenarse credenciales en el código fuente, se deben usar variables de entorno.
- Se debe aplicar validación y sanitización de entradas de usuario para evitar ataques.

Buenas prácticas

Para garantizar un desarrollo eficiente, escalable y mantenible, es fundamental seguir una serie de buenas prácticas que optimicen el rendimiento, la seguridad y la calidad del código.

A continuación, se presentan algunas recomendaciones claves que deben aplicarse en el desarrollo de software, especialmente en el manejo de bases de datos y la optimización de consultas:

- Usar índices correctamente en las bases de datos relacionales y no relacionales para mejorar el rendimiento de las consultas SQL.
- Hacer uso de la sentencia JOIN cuando se requiera hacer consultas a múltiples tablas.
- Evitar el uso de subconsultas en tablas que contienen gran cantidad de información.
- Evitar seleccionar todos los campos de una tabla si no serán utilizados en su totalidad.
- Eliminar sentencias JOIN innecesarias.
- Valorar la desnormalización para mejorar el rendimiento, es decir, agregar identificadores en tablas, aunque no exista una relación formal entre ellas.

- Considerar el particionamiento de tablas que llegarán a contener volúmenes extremadamente grandes de datos.
- Al realizar una sentencia JOIN, comenzar por la tabla que contendrá mayor cantidad de datos.
- Utilizar la sentencia JOIN en lugar de WITH cuando se necesite filtrar y agregar condiciones con múltiples tablas, para evitar la generación de numerosas consultas y subconsultas.
- Evaluar el uso de procedimientos almacenados cuando se requieran procesos complejos de acceso a la base de datos.
- Implementar mecanismos de caché cuando sea necesario.
- Reutilizar código; si una función o método puede ser útil en varias partes del sistema, es una buena práctica extraerlo y colocarlo en un trait para evitar duplicidad de código.
- Añadir comentarios a los métodos que realizan operaciones complejas o que implementan lógica difícil de comprender a simple vista.
- Utilizar transacciones cuando se requiera realizar inserciones en múltiples tablas.
- Evitar la carga diferida (lazy loading) cuando no sea estrictamente necesaria.

Plantillas de desarrollo

Estándares para uso de plantilla en Laravel

Versiones

- **Versión de Laravel:** 11.31
- **Versión de PHP:** 8.2
- **Versión de Composer:** 2.8.5
- **Gestor de base de datos relacional:** PostgreSQL 17.0
- **Gestor de base de datos no relacional:** MongoDB 8.0.5

Arquitectura y estructura del código

Rutas

Para la definición de rutas, se creará un archivo independiente por módulo, esto permite mantener una estructura más organizada y facilita el mantenimiento del código.

Controladores

Los controladores se definirán siguiendo la convención NombreController, por ejemplo, RoleController, la lógica de negocio se manejará a través de servicios (Service), manteniendo los controladores enfocados únicamente en la gestión de las solicitudes y respuestas.

Servicios

Los servicios serán responsables de gestionar toda la lógica de negocio, manteniendo los controladores ligeros y enfocados en la interacción con las solicitudes y respuestas.

Repositorios (opcional)

Si se opta por usar repositorios, estos encapsulan el acceso a la base de datos, para esto se deberá usar una interfaz para definir el contrato del repositorio:

- Se define un repositorio por cada entidad, siguiendo la convención NombreRepository, por ejemplo, RoleRepository.
- Los métodos del repositorio manejan todas las operaciones relacionadas con la base de datos, como consultas y manipulaciones de registros.

Interfaces (requerido si se usan repositorios)

Las interfaces definen los contratos que deben cumplir los repositorios, estableciendo los métodos que estos deben implementar:

- Se define una interfaz por cada repositorio, siguiendo la convención NombreRepositorioInterface, por ejemplo, RoleRepositoryInterface.
- Cada interfaz debe tener cada uno de los métodos que se implementaran en el repositorio.
- El flujo recomendado es:
    1. **Controlador**: recibe la solicitud y delega la lógica de negocio al servicio.
    2. **Servicio**: procesa la solicitud y utiliza el repositorio para interactuar con la base de datos.
    3. **Repositorio** (opcional): maneja las consultas y modificaciones en la base de datos.
    4. **Interfaz** (requerido si se usa repositorios): asegura que el repositorio cumpla con una estructura definida, facilitando cambios en la implementación futura.

Jobs

Se utilizarán Jobs para manejar tareas que requieran un tiempo de ejecución prolongado, permitiendo procesarlas en segundo plano sin afectar la experiencia del usuario ni el rendimiento del sistema.

Patrón de diseño Observer

Laravel proporciona el patrón de diseño Observer, el cual permite ejecutar acciones automáticamente después de eventos como la creación, actualización

o eliminación de registros. En este caso, se utiliza para registrar en una bitácora las actividades realizadas por los usuarios en el sistema, asegurando un mejor seguimiento y auditoría.

Traits

En la plantilla, se emplean Traits para definir métodos reutilizables en todo el sistema. Por ejemplo, se ha definido una función que se utilizará para paginar resultados, lo que permitirá estandarizar y reutilizar código en cualquier parte del sistema.

Route Model Binding

Se debe utilizar el Route Model Binding para obtener información de un modelo de manera eficiente y automática, evitando consultas manuales en el servicio.

Clase para manejar respuesta

Se ha definido una clase especializada para gestionar las respuestas enviadas al usuario desde los servicios, esta clase centraliza la generación de respuestas estándar, garantizando coherencia en los mensajes y códigos de estado HTTP.

Cuenta con métodos predefinidos como:

- **responseSuccess (estado HTTP 200 o 201)**: para respuestas exitosas.
- **responseConflict (estado HTTP 409)**: para conflictos en la solicitud.
- **responseBadRequest (estado HTTP 400)**: la solicitud contiene errores o parámetros inválidos.
- **responseUnprocessableEntity (estado HTTP 422)**: los datos enviados no pueden ser procesados.
- **responseNotFound (estado HTTP 404)**: el recurso solicitado no existe.
- **responseUnauthorized (estado HTTP 401)**: no autorizado.
- **responseForbidden (estado HTTP 403)**: no tiene permisos.

Gestor de base de datos relacional

Se recomienda el uso de PostgreSQL como gestor de base de datos. Ya que, incorpora mejoras significativas en rendimiento y funcionalidades que optimizan el desarrollo de aplicaciones.

Gestor de base de datos no relacional

La plantilla utiliza MongoDB como base de datos NoSQL para gestionar información clave del sistema, dicha base de datos se usa para:

- **Almacenamiento de sesiones de usuario**, permitiendo un acceso rápido y eficiente.
- **Registro de logs del sistema**, facilitando la auditoría y el monitoreo de eventos.
- **Seguimiento de la actividad del usuario**, asegurando un historial detallado de acciones.

Se recomienda mantener esta implementación para optimizar el rendimiento y la escalabilidad del sistema.

Comentar el código

Se recomienda incluir comentarios descriptivos en el código para explicar la funcionalidad de cada método, esto nos ayudará a la comprensión del código, los comentarios claros y concisos son útiles para cualquier desarrollador que trabaje en el proyecto a largo plazo.

Los comentarios deben realizarse utilizando **PHPDoc**, esto nos permitirá documentar correctamente las firmas de los métodos, los tipos de parámetros y los valores de retorno, mejorando la legibilidad y la comprensión del código.

Los comentarios son obligatorios cuando se cumplan los siguientes requisitos:

- **El método tiene una complejidad alta:**

  - Si el método realiza operaciones complejas o lógica difícil de entender a simple vista.
  - Integraciones con APIs externas.
- **El método tiene un propósito no evidente:**
  - Si el nombre del método no es suficientemente descriptivo o su funcionalidad no es clara.
- **El método tiene dependencias externas o efectos secundarios:**
  - Si el método depende de servicios externos, bases de datos, archivos, o si modifica estados globales.

Documentación de Endpoints

La plantilla utiliza Swagger para documentar las rutas de la API, facilitando la comprensión e interacción con los Endpoints, para esto se ha implementado la librería Scramble Laravel, que permite documentar automáticamente los endpoints API basándose en las definiciones de rutas y controladores.

Sin embargo, su uso **no es obligatorio**, cada equipo que implemente la plantilla es libre de utilizar su propia librería o método de documentación según sus necesidades y preferencias.