Taller Final

Julio Jiménez García
Juan Manuel Amaya Cadavid

Aseguramiento de la calidad
Carlos Andrés Jaramillo Patiño

Instituto Tecnológico Metropolitano
Facultad de ingeniería
Ingeniería de sistemas
2025

Introducción

El presente informe documenta el proceso de Aseguramiento de la Calidad (SQA) aplicado al sistema de software Kimai (https://github.com/kimai/kimai), una solución de seguimiento de tiempo (Time-Tracking) de código abierto basada en la web. Kimai es una aplicación de nivel empresarial diseñada para registrar horas de trabajo, gestionar presupuestos por proyecto y generar facturación, utilizada actualmente por una amplia comunidad global debido a su arquitectura modular y escalable.

Objetivo General

El objetivo principal de este caso práctico es aplicar de manera integrada las estrategias, técnicas y herramientas de validación y verificación de software adquiridas durante el curso. Se busca demostrar la capacidad para planificar, ejecutar y documentar un ciclo completo de pruebas (STLC), abarcando desde pruebas funcionales manuales y automatizadas hasta análisis estático de código y evaluaciones de seguridad bajo estándares de industria.

Justificación de la Elección

Se seleccionó el repositorio de Kimai por las siguientes razones técnicas:

1. **Robustez Arquitectónica:** Al estar construido sobre el framework Symfony (PHP), ofrece una estructura de patrones de diseño claros (MVC, Inyección de Dependencias) ideales para el análisis de mantenibilidad y deuda técnica.

2. **Entorno Contenerizado:** La disponibilidad de despliegue mediante Docker permite aislar el entorno de pruebas, facilitando la ejecución de pruebas de integración y sistema sin interferencias externas.

3. **Relevancia de Negocio:** Al ser un sistema que gestiona datos sensibles (facturación y tiempos laborales), permite diseñar casos de prueba críticos orientados a la seguridad (OWASP) y la integridad de los datos, cumpliendo con la exigencia de utilizar un "proyecto real".

Contexto del Proyecto: Descripción Funcional

Kimai funciona como un sistema SaaS (Software as a Service) multiusuario que permite la trazabilidad de actividades laborales. Sus módulos principales incluyen:

- Gestión de Entidades: Administración jerárquica de Clientes, Proyectos y Actividades.
- Registro de Tiempos (Timesheet): Interfaz de cronómetro en tiempo real y entrada manual de registros de duración definida.
- Reportes y Facturación: Generación de informes detallados exportables (PDF/Excel/HTML) y creación de facturas basadas en tarifas configurables.
- Control de Acceso (RBAC): Gestión de permisos basada en roles (Usuario, Jefe de Equipo, Administrador).

Arquitectura General

El sistema implementa una arquitectura Monolítica Modular basada en el patrón Modelo-Vista-Controlador (MVC) .

- Capa de Presentación (Vista): Renderizado del lado del servidor utilizando el motor de plantillas Twig y componentes interactivos en JavaScript (Webpack Encore).
- Capa de Lógica de Negocio (Controlador): Gestionada por el núcleo de Symfony, manejando el enrutamiento, la autenticación y la validación de datos.
- Capa de Datos (Modelo): Utiliza Doctrine ORM para la abstracción de base de datos, permitiendo la persistencia en sistemas relacionales.

Tecnologías Utilizadas

- Backend: PHP 8.1+ (Framework Symfony).
- Frontend: HTML5, CSS3 (Bootstrap), JavaScript.
- Base de Datos: MySQL / MariaDB (Desplegada en contenedor Docker sqldb).
- Infraestructura: Docker y Docker Compose para la orquestación de servicios.
- Servidor Web: Apache/Nginx.

Alcance Funcional Cubierto por las Pruebas

Las actividades de aseguramiento de calidad se limitarán al flujo crítico de negocio ("Core Business Loop"):

1. Autenticación y autorización de usuarios.
2. Ciclo de vida de administración: Creación de Clientes y Proyectos.
3. Ciclo de vida operativo: Registro, edición y eliminación de entradas de tiempo.
4. Validación de seguridad en formularios de entrada.

Requerimientos y Criterios de Calidad

Requisitos Funcionales (RF)

Se han identificado los siguientes requisitos esenciales para la validación:

- RF-01 Gestión de Sesiones: El sistema debe permitir el ingreso solo a usuarios registrados con credenciales válidas y bloquear el acceso tras intentos fallidos reiterados.
- RF-02 Jerarquía de Datos: No se debe permitir la creación de una "Actividad" sin la asignación previa a un "Proyecto", ni un "Proyecto" sin un "Cliente" asociado.
- RF-03 Registro de Tiempos: El sistema debe calcular automáticamente la duración basada en la hora de inicio y fin, impidiendo registros con fechas futuras o duraciones negativas.
- RF-04 Exportación: El sistema debe permitir generar exportaciones de datos en formatos legibles (PDF, CSV) filtrados por rangos de fechas.

Requisitos No Funcionales (RNF)

- RNF-01 Seguridad: Todas las contraseñas deben almacenarse utilizando algoritmos de hash robustos (Bcrypt/Argon2). El sistema debe proteger contra ataques XSS en los campos de descripción.
- RNF-02 Mantenibilidad: El código fuente debe cumplir con los estándares PSR-12 de PHP y mantener una deuda técnica controlada (Calificación A/B en SonarQube).
- RNF-03 Portabilidad: El sistema debe ser capaz de ejecutarse en cualquier entorno compatible con contenedores Docker.

Criterios de Aceptación

Para considerar una funcionalidad como "aprobada", debe cumplir con:

- Ejecución Exitosa: El resultado real coincide con el resultado esperado en el caso de prueba.
- Persistencia: Los datos creados deben ser visibles y recuperables tras reiniciar la sesión.
- Sin Errores Críticos: La ejecución no debe generar códigos de respuesta HTTP 500 ni excepciones no controladas en la interfaz de usuario.

Atributos de Calidad Analizados

Basado en el estándar ISO/IEC 25010 y el análisis estático realizado:

- Funcionalidad (Adecuación): Verificada mediante pruebas manuales del flujo de trabajo principal.
- Mantenibilidad: Evaluada mediante SonarQube, analizando métricas de complejidad ciclomática, duplicidad de código y "Code Smells".
- Seguridad: Evaluada mediante revisión de vulnerabilidades (OWASP Top 10) y análisis de puntos calientes (Security Hotspots) en el código fuente.

- Usabilidad: Verificada mediante la consistencia de la interfaz gráfica y la validación de formularios amigables al usuario.

Plan de Pruebas

Introducción y Alcance

El presente Plan de Pruebas define la estrategia, los recursos y el cronograma para validar la calidad del software Kimai. El enfoque se centra en verificar que el sistema cumpla con los requisitos funcionales y no funcionales descritos anteriormente, asegurando un despliegue libre de defectos críticos.

El alcance incluye:

- Módulos Funcionales: Login, Gestión de Clientes, Proyectos, Actividades y Registro de Tiempos (Timesheet).
- Calidad del Código: Análisis estático de la totalidad del código fuente PHP.
- Seguridad: Análisis dinámico de vulnerabilidades web básicas.

Fuera del alcance:

- Pruebas de pasarelas de pago (plugins externos).
- Pruebas de estrés a nivel de infraestructura de red.

Niveles de Prueba

Se abordarán los siguientes niveles para garantizar una cobertura integral:

1. Pruebas Unitarias: Verificación de la lógica interna mediante el análisis de los tests existentes en el repositorio (PHPUnit) y métricas de cobertura reportadas por SonarQube.
2. Pruebas de Integración: Validación de la comunicación entre el contenedor de la aplicación (kimai-app) y el contenedor de base de datos (sqldb), asegurando la persistencia de datos.
3. Pruebas de Sistema: Ejecución de flujos completos de negocio (End-to-End) simulando el comportamiento de un usuario real a través de la interfaz web.
4. Pruebas de Aceptación: Validación final contra los criterios de aceptación definidos en los requerimientos funcionales.

Estrategias y Técnicas de Prueba

Pruebas Funcionales (Caja Negra)

Se utilizará una estrategia de Caja Negra, interactuando con la interfaz gráfica sin manipular el código interno durante la ejecución.

- Partición de Equivalencia: Para validar entradas en formularios (ej. fechas válidas vs. inválidas).
- Análisis de Valores Límite: Para campos numéricos como tarifas y duraciones de tiempo.
- Tablas de Decisión: Para validar la lógica de roles (Admin vs. Usuario) y permisos de acceso.

Pruebas de Análisis Estático (Caja Blanca)

Se empleará una estrategia de Caja Blanca automatizada para medir la salud del código.

- Herramienta: SonarQube Community Edition.
- Métricas a evaluar: Deuda técnica, Code Smells, Bugs potenciales y duplicidad de código.

Pruebas de Seguridad (DAST)

Se realizará un análisis dinámico de seguridad (Dynamic Application Security Testing).

- Herramienta: OWASP ZAP (Zed Attack Proxy).
- Objetivo: Identificar vulnerabilidades del OWASP Top 10, específicamente Inyección SQL, XSS (Cross-Site Scripting) y exposición de datos sensibles en cabeceras HTTP.

Pruebas Automatizadas

Se automatizarán los flujos repetitivos y críticos para pruebas de regresión (Smoke Testing).

- Herramienta: Selenium WebDriver (Python)
- Alcance: Login y Creación básica de registros.

Entorno y Herramientas

Para la ejecución del plan se ha configurado el siguiente entorno de laboratorio:

Recurso / Herramienta	**Uso Específico**	**Versión / Detalle**

Docker & Docker Compose	Orquestación del entorno de pruebas	Contenedores Linux (Debian)
Kimai Repository	SUT (System Under Test)	v2 (Apache image)
SonarQube & Scanner	Análisis Estático de Calidad	Community Edition + CLI Docker
OWASP ZAP	Escaneo de Vulnerabilidades	Versión Estable
Selenium WebDriver (Python)	Automatización de pruebas funcionales	
Google Chrome / DevTools	Ejecución manual y debug	Última versión estable

Criterios de Entrada y Salida

Criterios de Entrada (Start Criteria):

- El entorno Docker debe estar desplegado y accesible vía http://localhost:8001.
- La base de datos debe estar poblada con datos maestros mínimos (1 usuario admin).
- El servidor SonarQube debe estar activo en el puerto 9000.

Criterios de Salida (Exit Criteria):

- Ejecución del 100% de los casos de prueba manuales definidos.
- Ejecución exitosa de los scripts automatizados.
- Resolución o documentación de cualquier defecto bloqueante (Severidad Crítica).
- Generación del reporte de métricas de SonarQube.

Cronograma de Actividades

Fase	Actividad	Herramienta	Duración Est.
1. Planificación	Configuración del entorno Docker y clonado de repo	Docker / Git	2 Horas
2. Diseño	Redacción de casos de prueba (Matriz)	Documentación	3 Horas
3. Ejecución	Pruebas Manuales Funcionales	Web Browser	2 Horas
4. Automatización	Grabación y depuración de scripts	Selenium IDE	2 Horas
5. Análisis Técnico	Escaneo de Código y Seguridad	SonarQube / ZAP	2 Horas
6. Cierre	Consolidación de hallazgos e informe final	Editor de texto	2 Horas

Diseño de Casos de Prueba

Estrategia de Diseño

Para el diseño de los casos de prueba se han aplicado las siguientes técnicas de caja negra, conforme a los requisitos del aseguramiento de calidad:

- Partición de Equivalencia: Se dividieron los datos de entrada en clases válidas (datos correctos) e inválidas (formatos erróneos) para reducir el número total de pruebas necesarias.
-

- Análisis de Valores Límite: Se diseñaron casos para verificar el comportamiento del sistema en los bordes de los rangos aceptados (ej. fechas futuras, duraciones negativas).
- Pruebas de Seguridad (Input Validation): Se incluyeron vectores de ataque básicos (XSS) en campos de texto para validar la sanitización de entradas.
- Tablas de Decisión: Utilizadas para verificar la lógica de permisos según el rol del usuario (Administrador vs. Usuario Estándar).

Matriz de Casos de Prueba

A continuación, se detallan los casos de prueba diseñados para el ciclo de validación actual:

Bloque A: Pruebas Funcionales y de Lógica (CP-01 a CP-05)

Atributo / Caso	**CP-01**	**CP-02**	**CP-03**	**CP-04**	**CP-05**
Módulo	Autenticación	Autenticación	Clientes	Tiempos	Tiempos
Objetivo	Verificar ingreso con credenciales válidas.	Verificar bloqueo ante pass incorrecta.	Validar creación de nuevo cliente.	Validar restricción de fechas futuras.	Validar cálculo de duración negativa.
Técnica	Camino Feliz	Partición Equivalencia	Funcionalidad Básica	Valores Límite	Lógica de Negocio
Res. Esperado	Redirección al Dashboard sin errores.	Acceso denegado. Mensaje de error.	Cliente guardado y visible en lista.	Bloqueo o advertencia de fecha inválida.	Error: "Fecha final debe ser posterior".

Bloque B: Seguridad, Integridad y Usabilidad (CP-06 a CP-10)

Atributo / Caso	CP-06	CP-07	CP-08	CP-09	CP-10
Módulo	Seguridad	Proyectos	Seguridad	Exportación	Interfaz
Objetivo	Verificar protección contra XSS.	Validar campos obligatorios.	Verificar acceso URL sin sesión.	Validar descarga de reporte PDF.	Validar cambio de idioma.
Técnica	Inyección Código	Tablas Decisión	Control Acceso	Funcionalidad Salida	Usabilidad
Res. Esperado	Sanitización de entrada (texto literal).	Mensaje de validación "Requerido".	Redirección forzada al Login.	Descarga de archivo .pdf legible.	Interfaz cambia textos a Español.

Ejecución y Evidencias de Pruebas

En esta sección se documenta la ejecución de los casos de prueba diseñados, demostrando la conformidad del sistema Kimai con los requisitos funcionales y no funcionales. Se utilizaron herramientas de navegador (Chrome DevTools), automatización (Selenium IDE) y análisis de seguridad.

Ejecución de Pruebas Manuales (10 Casos)

A continuación, se presentan los resultados obtenidos tras la ejecución manual de los casos de prueba diseñados en la sección anterior.

CP-01: Autenticación Exitosa (Camino Feliz)

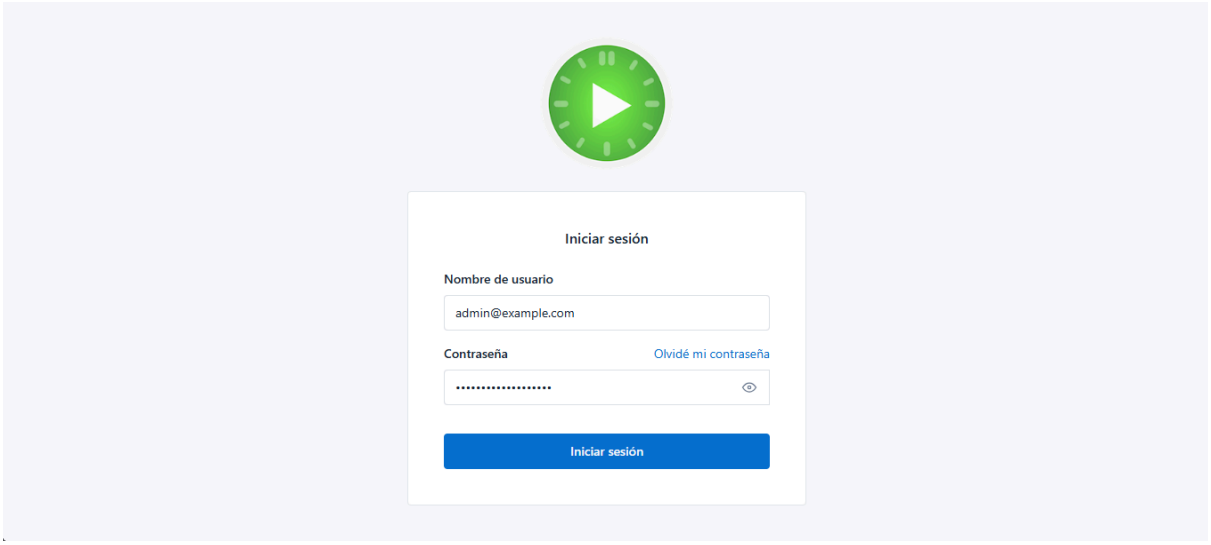
Instrucciones:

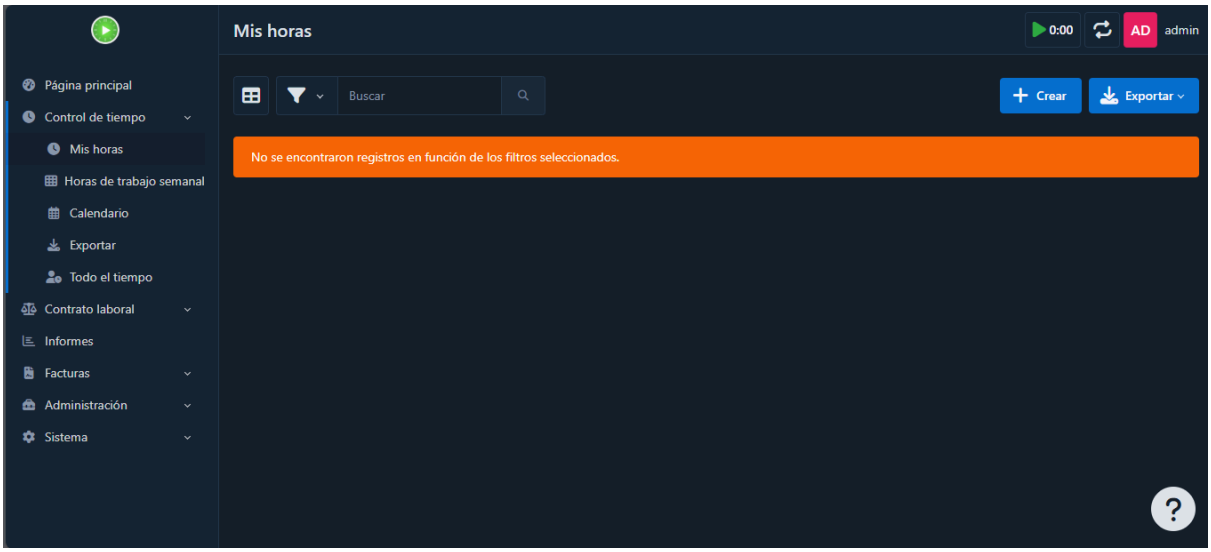
Navegar a http://localhost:8001.

Usuario: admin@example.com | Contraseña: TallerCalidad2025.

Clic en "Log in".

Evidencia:



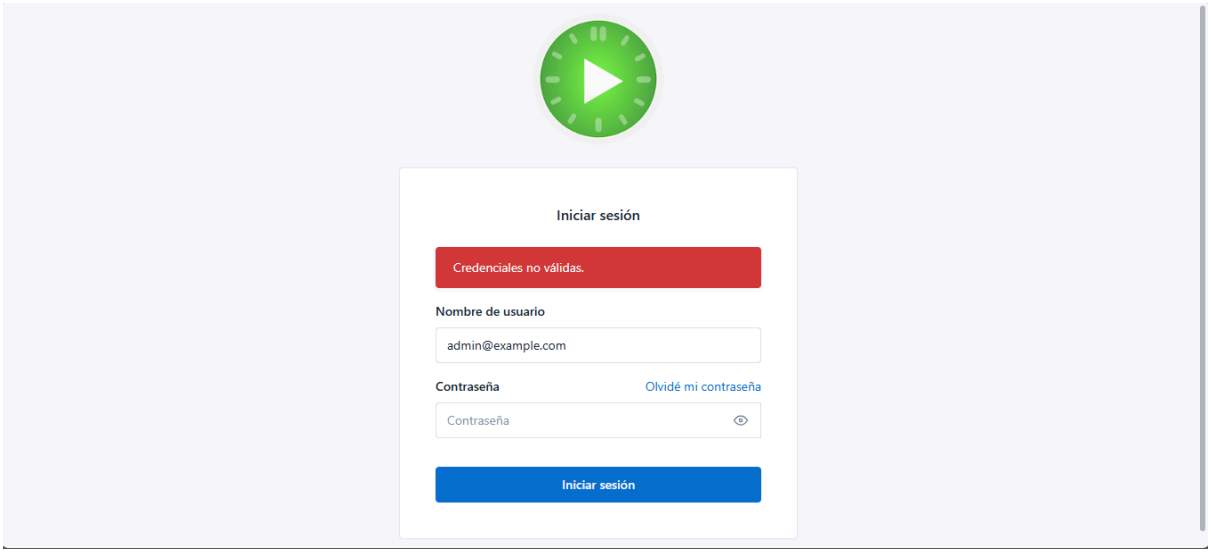


CP-02: Autenticación Fallida (Validación)

- **Instrucciones:**

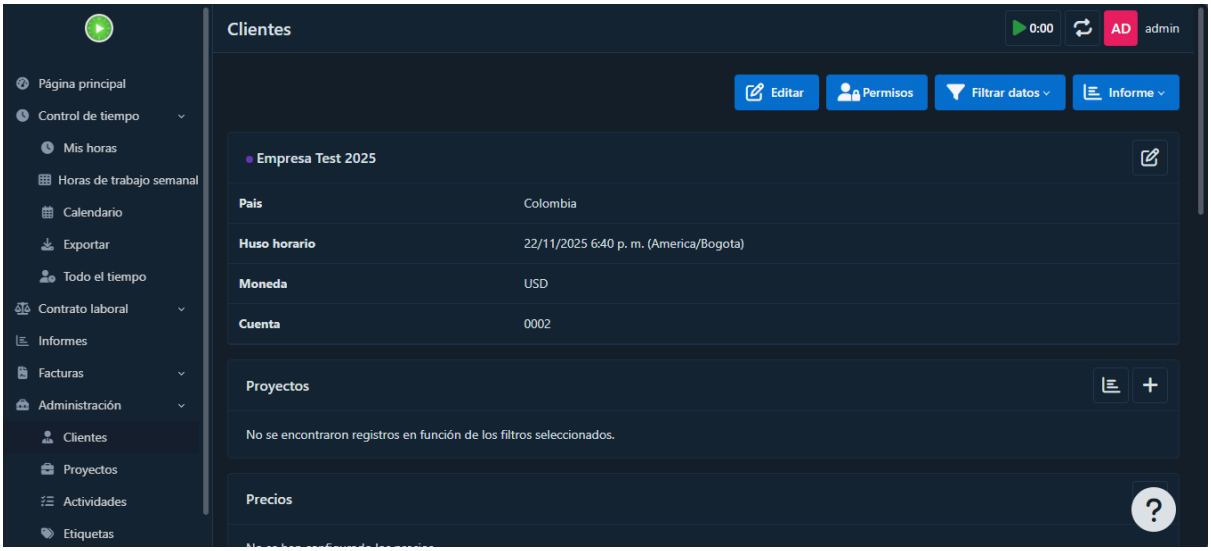
1. Cerrar sesión o abrir ventana incógnito.
2. Ingresar admin@example.com y contraseña Erronea123.

- **Evidencia:**



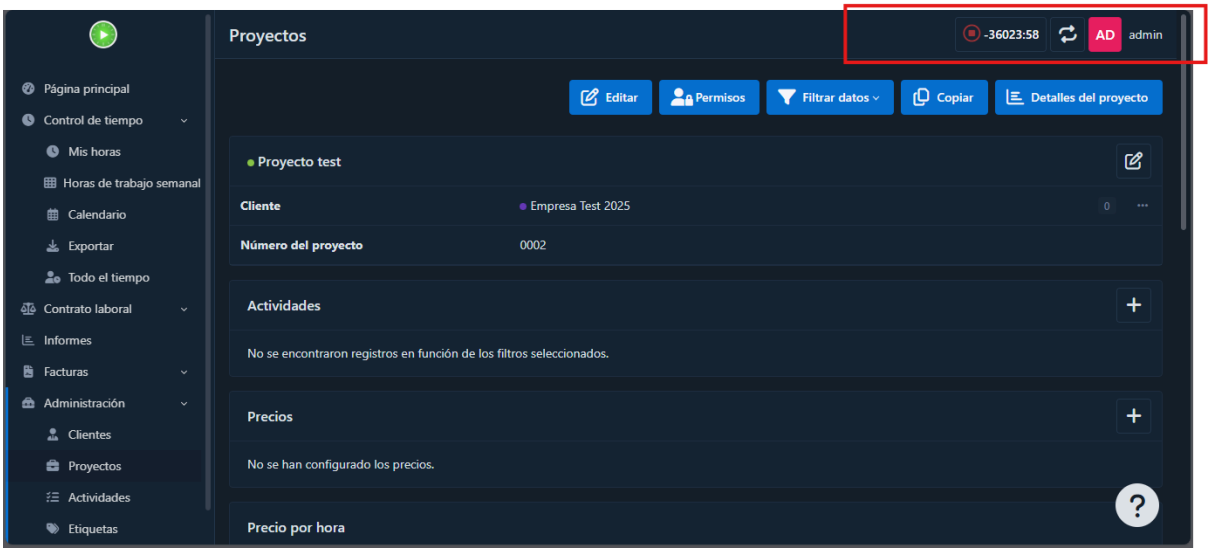
CP-03: Creación de Cliente (Funcional)

- **Instrucciones:**
  1. Menú Administración -> "Clientes" -> Botón azul "Crear".
  2. Nombre: "Empresa Test 2025".
  3. País: Colombia.
  4. Guardar.
- **Evidencia:**



CP-04: Validación de Fechas Futuras (Valores Límite)

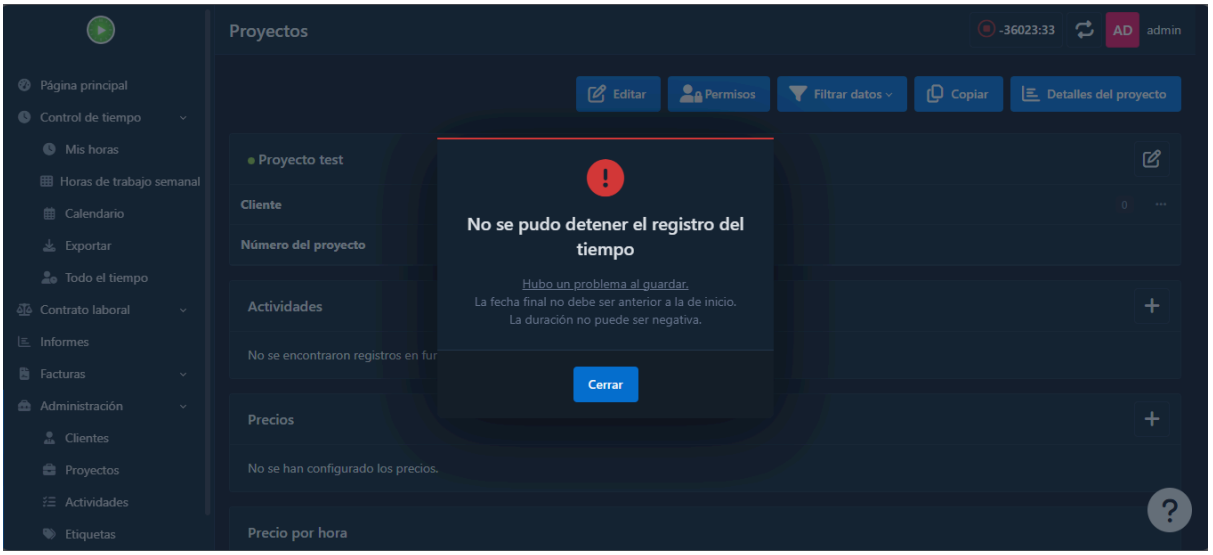
- **Instrucciones:**
    1. Clic en botón "Play" (arriba derecha) o crear registro de tiempo.
    2. Intentar poner una fecha de inicio del año 2030 (Ej: 2030-01-01).
    3. Intentar guardar.
- **Evidencia:**



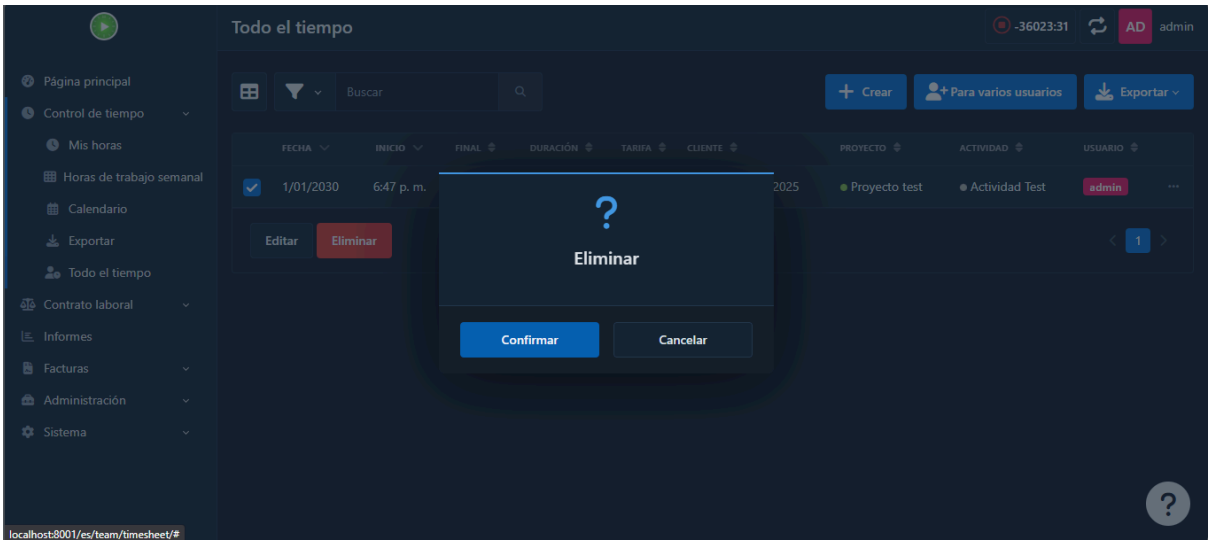
Resultado Real: FALLIDO (Defecto Encontrado).

Observación / Análisis:

El sistema permite INICIAR el registro con fecha futura, pero genera un error bloqueante al intentar DETENERLO por "duración negativa" El usuario queda con un registro imposible de cerrar

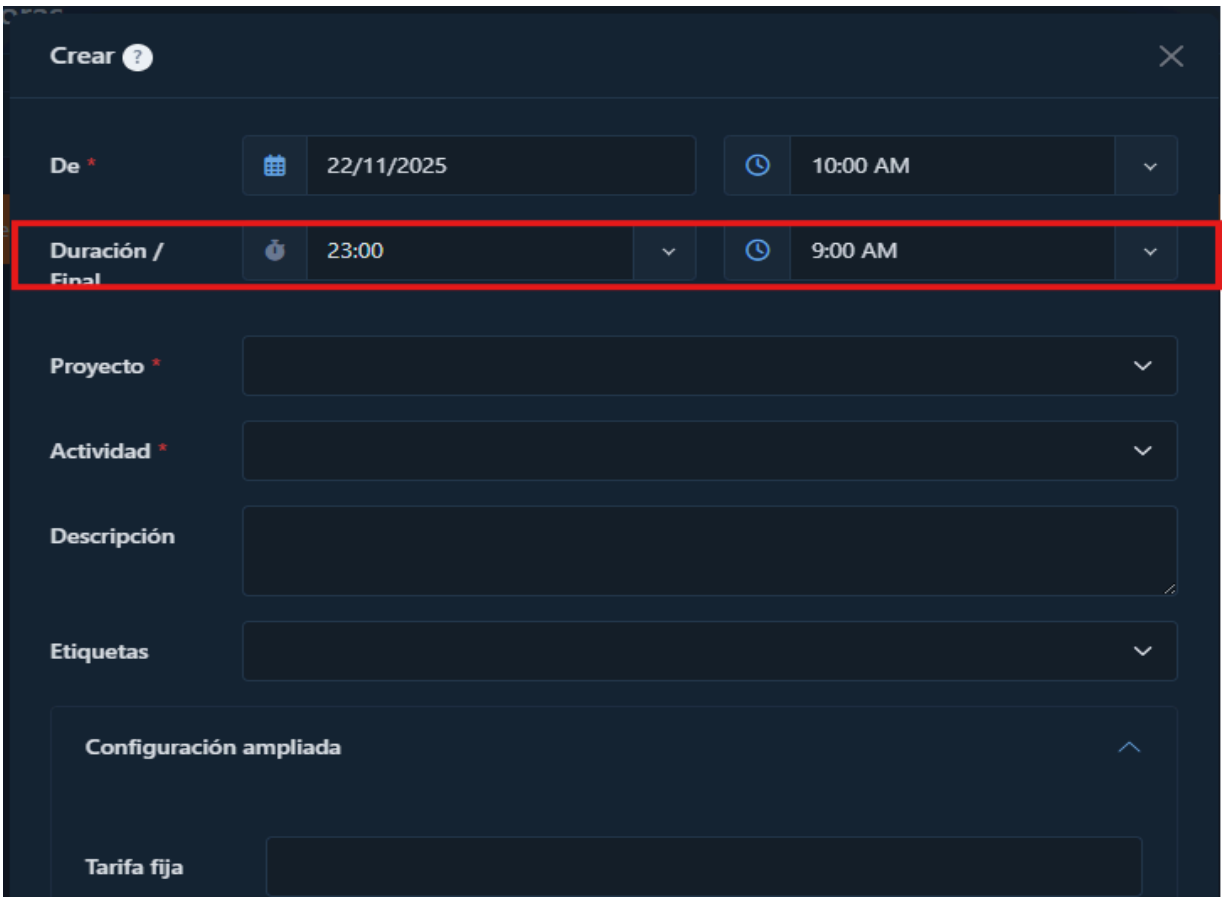


Recuperación: Se requiere intervención de un Administrador para forzar el borrado desde el panel "Todo el tiempo", ya que el usuario estándar queda bloqueado.



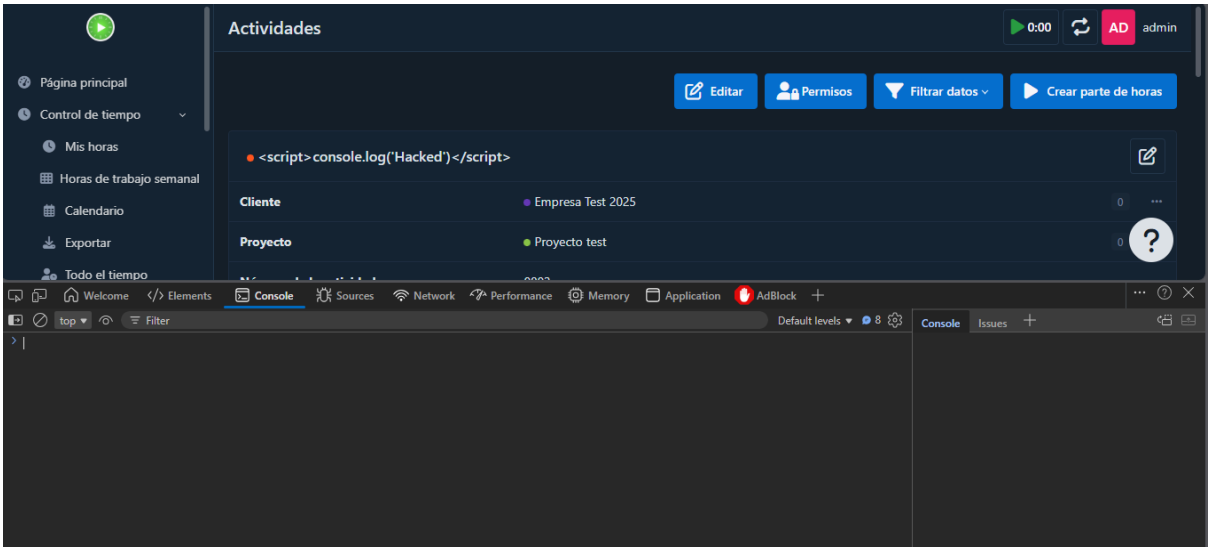
CP-05: Duración Negativa (Lógica de Negocio)

- Instrucciones:
    1. Crear registro de tiempo manual.
    2. Hora Inicio: 10:00.
    3. Hora Fin: 09:00 (Una hora antes).
- Evidencia: Exitoso: (Prevención Frontend). Los campos "Final" y "Duración" cambian dinámicamente (Ver Fig. 5). El frontend impide ingresar rangos negativos manualmente.



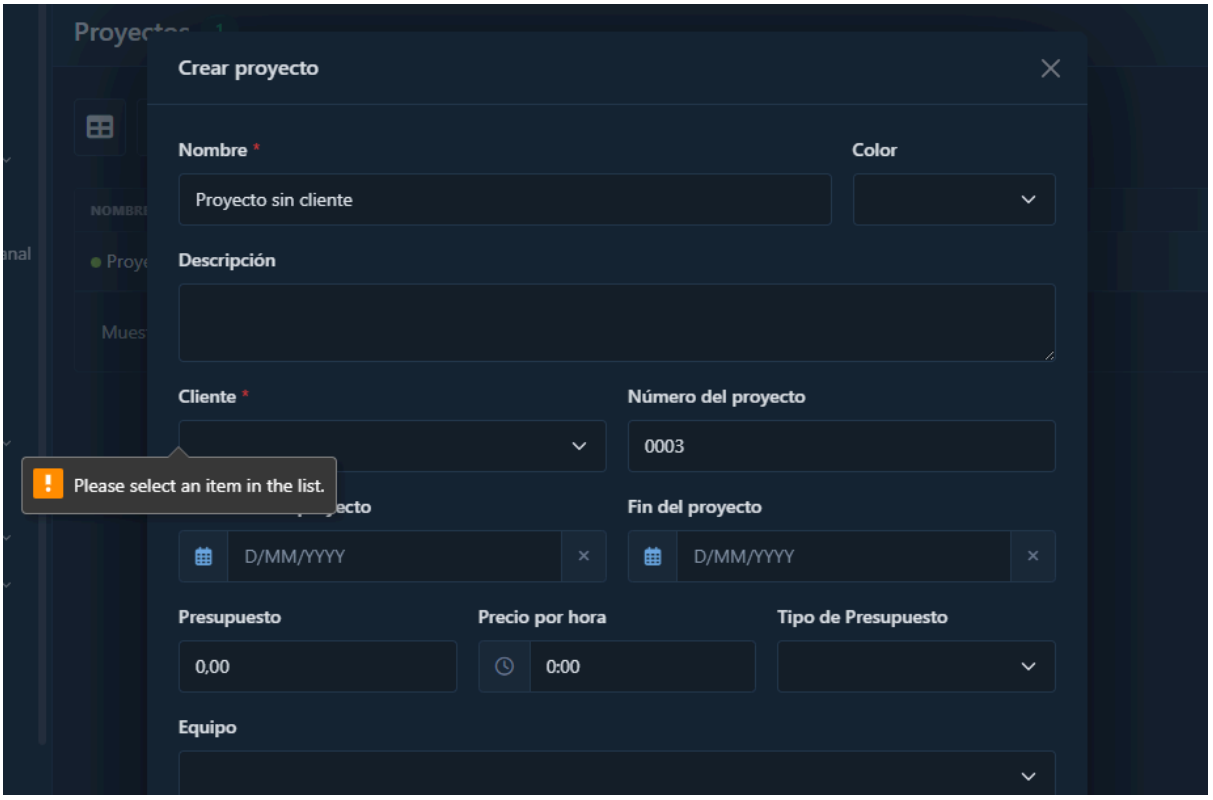
CP-06: Prueba de Seguridad XSS (OWASP)

- **Instrucciones:**
    1. Crear una nueva Actividad.
    2. En el nombre, pegar: <script>console.log('Hacked')</script>.
    3. Guardar y recargar la página.
- **Evidencia:** El sistema permite guardar la cadena, pero aplica *Output Encoding* al renderizarla. El script se muestra como texto plano y NO se ejecuta en la consola del navegador



CP-07: Campos Obligatorios (Integridad)

- **Instrucciones:**
    1. Crear un Proyecto.
    2. Dejar el campo "Cliente" vacío.
    3. Escribir nombre del proyecto y dar "Guardar".
- **Evidencia:**



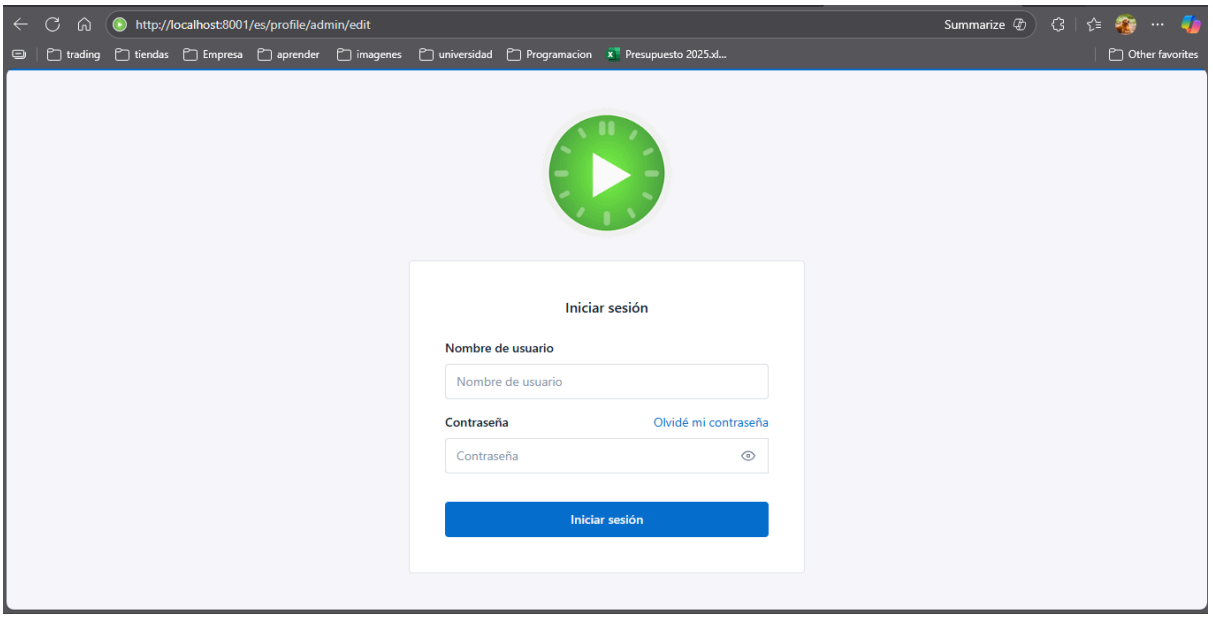
- Exitoso con Observaciones.

Funcionalmente correcto (bloquea el envío), pero presenta Defecto de Localización. El mensaje de validación ("Please select an item...") se muestra en Inglés a pesar de que la interfaz está en Español

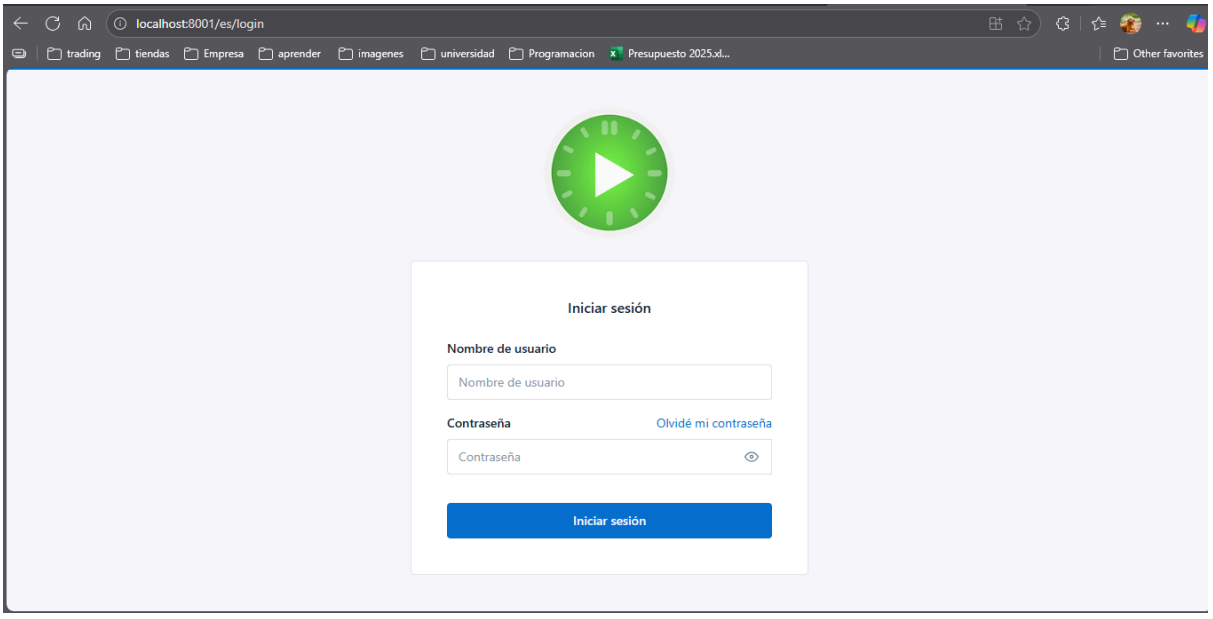
CP-08: Acceso No Autorizado (Broken Access Control)

- **Instrucciones:**
  1. Cerrar sesión (Logout).
  2. Pegar en la URL: http://localhost:8001/es/profile/admin/edit
- **Evidencia:**

URL lista



El sistema no permite y vuelve a la url de login

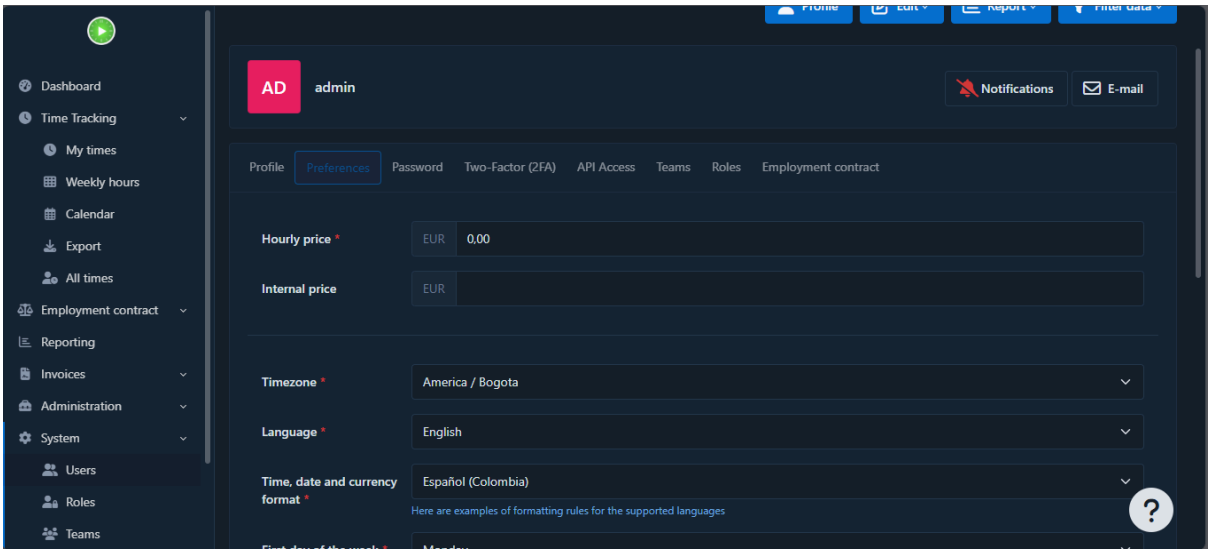


CP-09: Exportación de Datos (Salida)

- **Instrucciones:**
    1. Ir al menú "Exportar".
    2. Seleccionar "Este año".
    3. Clic en botón PDF
- **Evidencia:**

CP-10: Cambio de Idioma (Usabilidad)

- **Instrucciones:**
    1. Ir a Perfil (Icono usuario arriba derecha) -> Preferencias.
    2. Cambiar idioma a "English"
- **Evidencia:**



Ejecución de Pruebas Automatizadas (5 Casos)

Se importan las siguientes librerías además se están colocando las variables principales a utilizar con los valores inicializados.

```python
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException, NoSuchElementException, StaleElementReferenceException
import time
import uuid



# URL base del sistema Kimai
BASE_URL = "http://localhost:8001"
LOGIN_PATH = "/es/login"
# La ruta de destino después del login es típicamente el timesheet.
POST_LOGIN_BASE_PATH = "/es/timesheet/"
# Ruta de la lista de clientes (destino después de crear un cliente)
CUSTOMER_LIST_PATH = "/es/admin/customer/"

# Credenciales de prueba VÁLIDAS
TEST_USERNAME = "admin@example.com"
TEST_PASSWORD = "TallerCalidad2025"

# Selectores de Login
SELECTOR_USERNAME = "username"
SELECTOR_PASSWORD = "password"
```

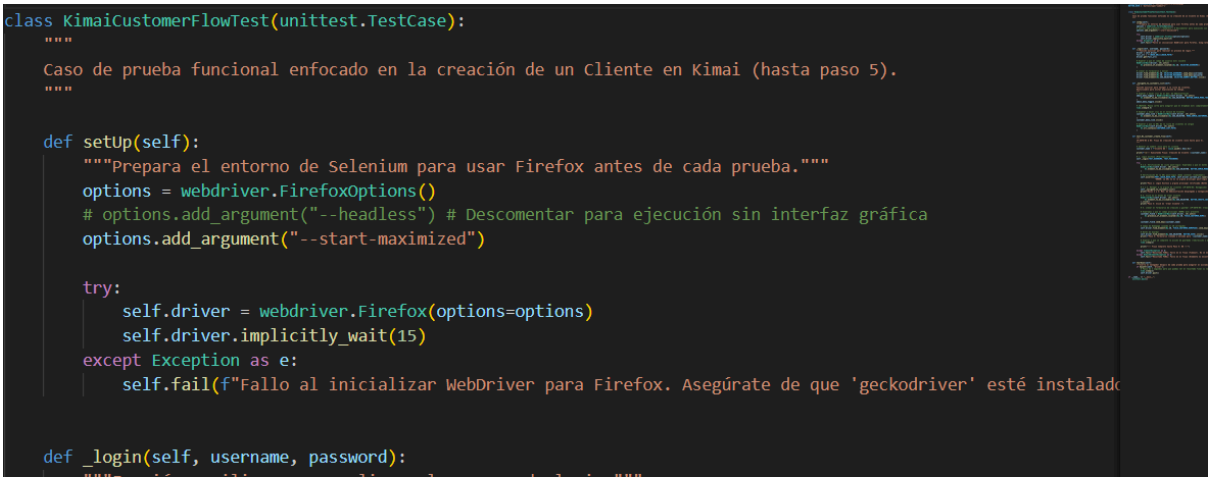
```python
    # Credenciales de prueba VÁLIDAS
    TEST_USERNAME = "admin@example.com"
    TEST_PASSWORD = "TallerCalidad2025"

    # Selectores de Login
    SELECTOR_USERNAME = "username"
    SELECTOR_PASSWORD = "password"
    SELECTOR_SUBMIT_BUTTON = 'button[type="submit"]'
    SELECTOR_SUCCESS_MESSAGE = '.alert-success'

    # Selectores de Administración y Clientes
    # Selector del botón principal "Administración" (para desplegar el menú)
    BUTTON_ADMIN_MENU_TOGGLE = 'a.navbar-menu-admin[data-bs-toggle="dropdown"]'
    # Selector de enlace en el submenú para ir a la lista de clientes
    MENU_ADMIN_CUSTOMERS_LINK = 'a.navbar-menu-customers[href*="/customer/"]'
    # Selector del botón de "Crear nuevo cliente"
    BUTTON_CREATE_CUSTOMER = 'a[href*="/customer/create"]'

    # ** IDs generados por Symfony para los campos **
    FIELD_CUSTOMER_NAME = 'customer_edit_form_name'
    FIELD_CUSTOMER_HOMEPAGE = 'customer_edit_form_homepage'
    BUTTON_SAVE = 'button[type="submit"]'
```

Para estos casos de pruebas automatizadas decidimos utilizar un navegador diferente al que se usó en las pruebas manuales, por lo que se utilizó firefox, se hace la preparación del entorno web (la opción que nos da selenium para abrir firefox) .



```
class KimaiCustomerFlowTest(unittest.TestCase):
    """
    Caso de prueba funcional enfocado en la creación de un Cliente en Kimai (hasta paso 5).
    """

    def setUp(self):
        """Prepara el entorno de Selenium para usar Firefox antes de cada prueba."""
        options = webdriver.FirefoxOptions()
        # options.add_argument("--headless") # Descomentar para ejecución sin interfaz gráfica
        options.add_argument("--start-maximized")

        try:
            self.driver = webdriver.Firefox(options=options)
            self.driver.implicitly_wait(15)
        except Exception as e:
            self.fail(f"Fallo al inicializar WebDriver para Firefox. Asegúrate de que 'geckodriver' esté instalad


    def _login(self, username, password):
```

CP-AUTO-01 (Login): Ingresa credenciales y accede al sistema.
primero se crea un método auxiliar que me identifica los campos a llenar además envía los valores al formulario.

```
def _login(self, username, password):
    """Función auxiliar para realizar el proceso de login."""
    driver = self.driver
    full_url = f"{BASE_URL}{LOGIN_PATH}"
    driver.get(full_url)

    # Esperar a que el campo de usuario esté visible
    WebDriverWait(driver, 15).until(
        EC.presence_of_element_located((By.ID, SELECTOR_USERNAME))
    )

    # Llenar el formulario y enviar
    driver.find_element(By.ID, SELECTOR_USERNAME).send_keys(username)
    driver.find_element(By.ID, SELECTOR_PASSWORD).send_keys(password)
    driver.find_element(By.CSS_SELECTOR, SELECTOR_SUBMIT_BUTTON).click()
```

Después simplemente desde el método principal se llama la función.

```
try:
    # 1.b. Verificación de éxito post-login: Esperamos a que el botón de Administración esté disponible.
    WebDriverWait(self.driver, 20).until(
        EC.element_to_be_clickable((By.CSS_SELECTOR, BUTTON_ADMIN_MENU_TOGGLE))
    )

    # Verificación de la URL post-login (Debe contener /timesheet/)
    self.assertIn(POST_LOGIN_BASE_PATH, self.driver.current_url.lower(),
                  "ERROR: La URL no es la página principal post-login (/timesheet/), el inicio de sesión

    print("Paso 1: Login Exitoso y página principal verificada (Botón Administración visible).")

    # 2 y 3. Navegar a la página de clientes (CP-AUTO-03: Navegación)
```

CP-AUTO-02 (Navegación): Accede al módulo de "Administrador" mediante selectores CSS.

```
def _navigate_to_customers_list(self):
    """
    Función auxiliar para navegar a la lista de clientes.
    Reutilizable para evitar duplicación de código.
    """
    # Esperar y hacer clic en el menú de Administración
    admin_menu_toggle = WebDriverWait(self.driver, 15).until(
        EC.element_to_be_clickable((By.CSS_SELECTOR, BUTTON_ADMIN_MENU_TOGGLE))
    )
    admin_menu_toggle.click()

    # CRÍTICO: Pausa corta para asegurar que el dropdown esté completamente desplega
    time.sleep(0.5)

    # Esperar y hacer clic en el enlace de Clientes
    customer_menu_link = WebDriverWait(self.driver, 15).until(
        EC.element_to_be_clickable((By.CSS_SELECTOR, MENU_ADMIN_CUSTOMERS_LINK))
    )
    customer_menu_link.click()

    # Esperar a que la URL de la lista de clientes se cargue
    WebDriverWait(self.driver, 15).until(
        EC.url_contains(CUSTOMER_LIST_PATH)
    )
```

una vez se loguea se procede hacer la navegacion.

```
# Verificación de la URL post-login (Debe contener /timesheet/)
self.assertIn(POST_LOGIN_BASE_PATH, self.driver.current_url.lower(),
              "ERROR: La URL no es la página principal post-login (/timesheet/), el inicio de sesión

print("Paso 1: Login Exitoso y página principal verificada (Botón Administración visible).")

# 2 y 3. Navegar a la página de clientes (CP-AUTO-03: Navegación)
self._navigate_to_customers_list()
print("Pasos 2 y 3: Menú de Administración desplegado y navegación a la lista de Clientes exitosa.")

# 4. Click en el botón de Crear Cliente
```

CP-AUTO-03 (Navegación): Accede al submódulo de "cliente" mediante selectores CSS.
se puede evidenciar en el CP-AUTO-02.
CP-AUTO-04 (Navegación): Clic en 'Crear Cliente'.
Una vez se hace la navegación para llegar a la página se hace uso del elemento que identifica

al botón que envía a la vista de creación.

```
# 4. Click en el botón de Crear Cliente
WebDriverWait(self.driver, 15).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, BUTTON_CREATE_CUSTOMER))
).click()
print("Paso 4: Click en 'Crear Cliente'.")
```

CP-AUTO-05 (Creación):Formulario llenado y enviado para 'Cliente.

```
    # 5. Llenar el formulario de creación y guardar (CP-AUTO-04: Creación)

    # Esperamos a que el campo principal (NAME) esté presente
    customer_field = WebDriverWait(self.driver, 15).until(
        EC.presence_of_element_located((By.ID, FIELD_CUSTOMER_NAME))
    )

    customer_field.send_keys(customer_name)

    # Campo de Homepage (usando el ID corregido)
    self.driver.find_element(By.ID, FIELD_CUSTOMER_HOMEPAGE).send_keys("http://www.ejemplo-qa.com")

    # Botón de Guardar
    self.driver.find_element(By.CSS_SELECTOR, BUTTON_SAVE).click()
    print(f"Paso 5: Formulario llenado y enviado para '{customer_name}'.")

    # Esperar a que se complete la acción de guardado (redirección o mensaje de éxito)
    time.sleep(2)

    print("--- Flujo Completo hasta Paso 5: OK ---")

except TimeoutException as e:
    self.fail(f"Resultado FINAL: Fallo en el flujo (Timeout). No se encontró el elemento a tiempo. URL act
except NoSuchElementException as e:
    self.fail(f"Resultado FINAL: Fallo en el flujo (Elemento no encontrado). Error: {e}")
```

ya por último hacemos el cierre de navegador y el llamado al método principal que ejecuta el flujo de pruebas.

```
    def tearDown(self):
        """Cierra el navegador después de cada prueba para asegurar el aislamiento."""
        if hasattr(self, 'driver'):
            # Pausa de 2 segundos para que puedas ver el resultado final si lo estás viendo en vivo
            time.sleep(2)
            self.driver.quit()

if __name__ == "__main__":
    unittest.main()
```

Pruebas de Rendimiento (Performance Testing)

Para evaluar la eficiencia del frontend de Kimai, se ejecutó una auditoría técnica utilizando Google Lighthouse. La prueba se realizó sobre la vista de "Timesheets" (/es/timesheet/), que representa una de las pantallas con mayor carga de interactividad y datos.

Escenario de Prueba:

Herramienta: Google Lighthouse (v10).

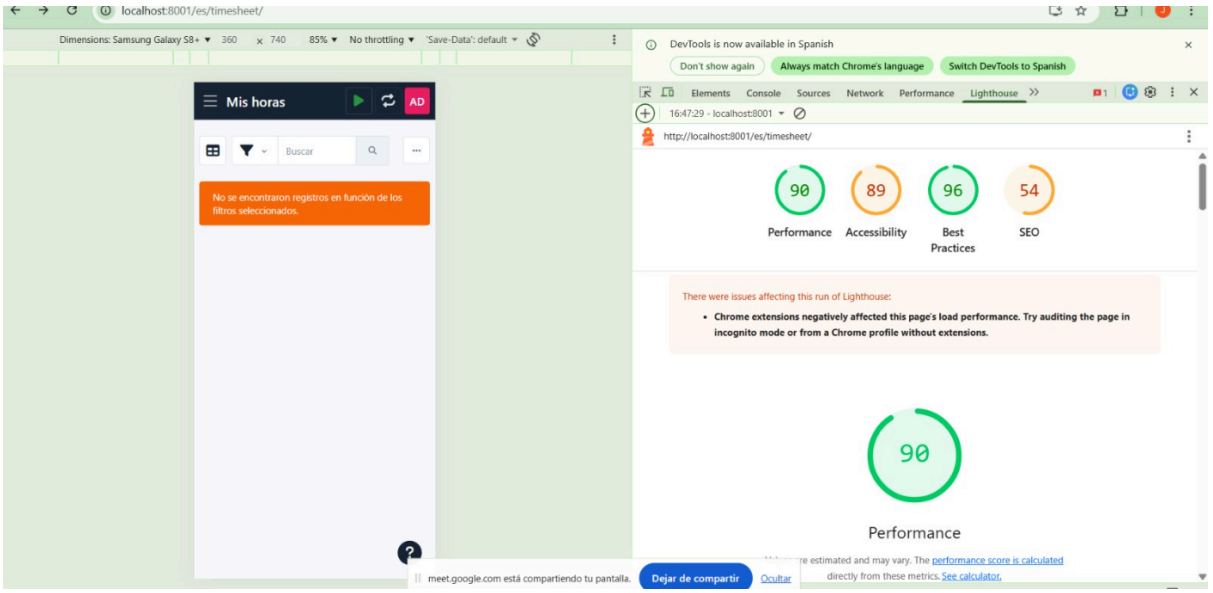
Entorno: Despliegue local en Docker (localhost:8001).

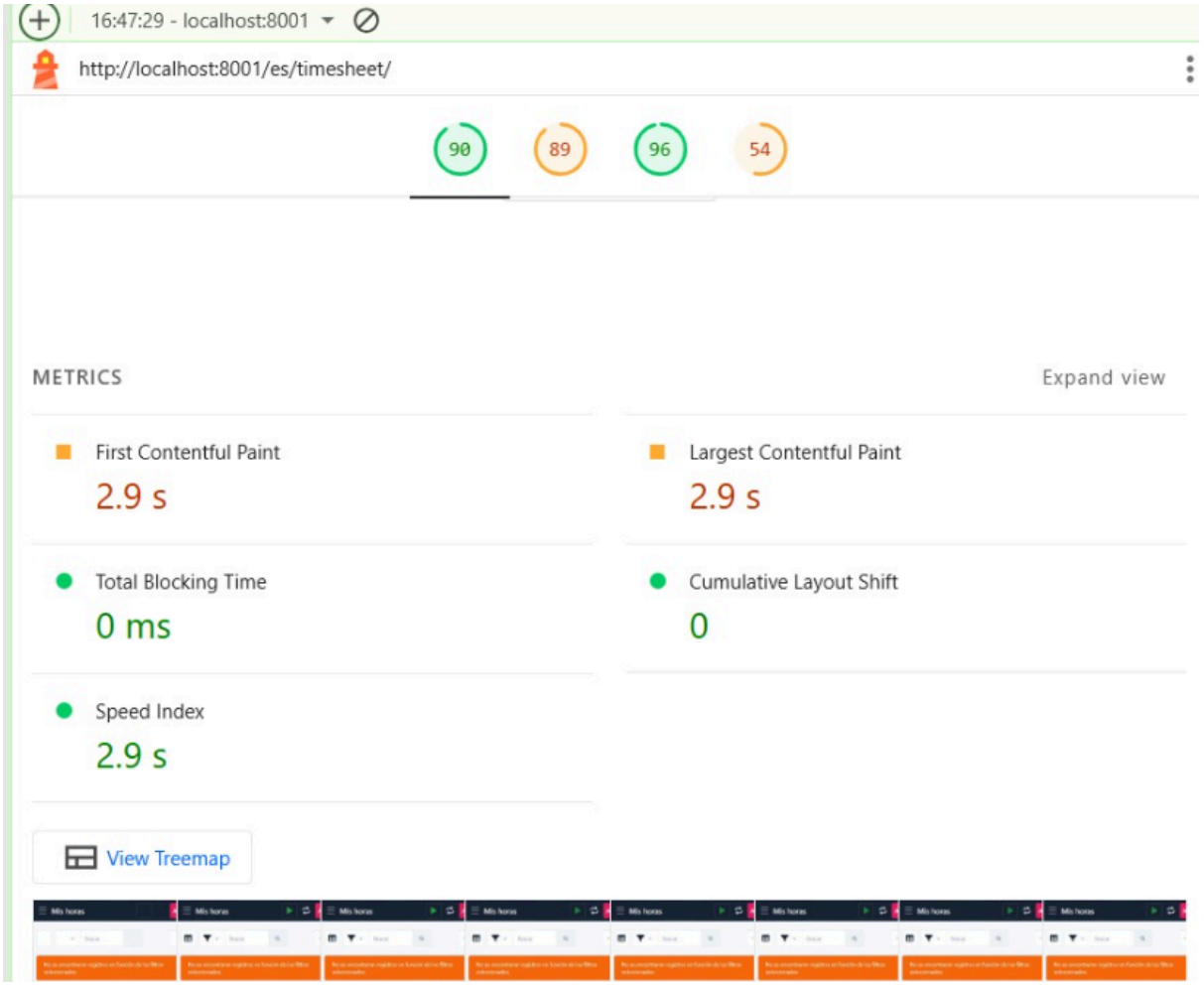
Configuración: Emulación de dispositivo de escritorio (Desktop).

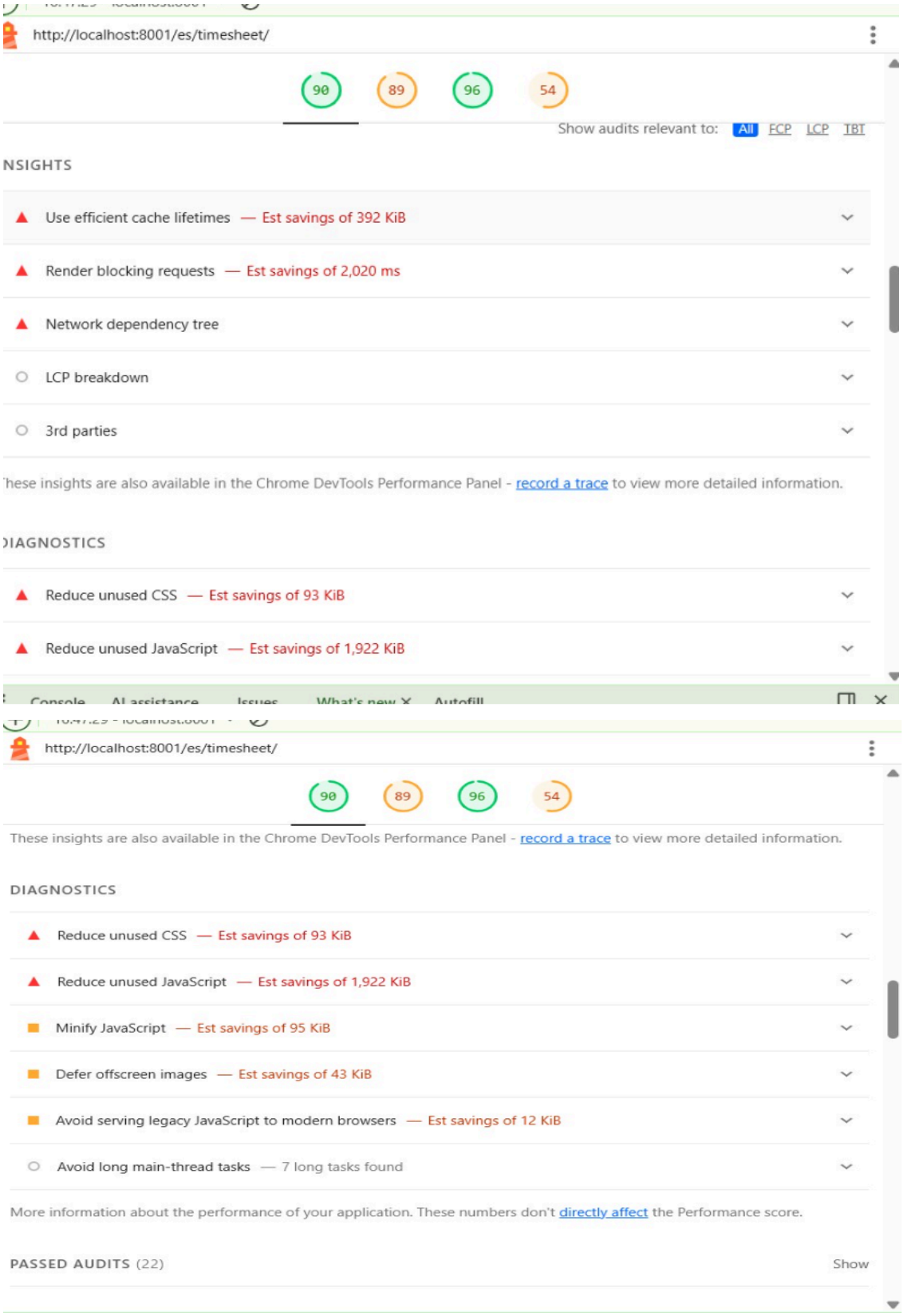
Resultados Obtenidos:

Métrica	Resultado	Estado	Análisis Técnico
Score de Rendimiento	**90 / 100**	🟢 Excelente	A pesar de la latencia en la carga inicial, la aplicación es altamente responsiva una vez renderizada.
FCP (First Contentful Paint)	**2.9 s**	🟠 Moderado	El usuario debe esperar casi 3 segundos para ver el primer contenido. Esto se debe a recursos que bloquean el renderizado.
LCP (Largest Contentful Paint)	**2.9 s**	🟠 Moderado	El elemento principal tarda lo mismo que el FCP, indicando que todo el contenido aparece "de golpe" tras la espera.

TBT (Total Blocking Time)	**0 ms**	🟢 Perfecto	No hay bloqueo del hilo principal del procesador. La interactividad es inmediata tras la carga.
CLS (Layout Shift)	**0**	🟢 Perfecto	Estabilidad visual total. Los elementos no se mueven inesperadamente mientras cargan.


localhost:8001/es/timesheet/
Mis horas
AD
Buscar
No se encontraron registros en función de los filtros seleccionados.
DevTools is now available in Spanish
Don't show again
Always match Chrome's language
Switch DevTools to Spanish
Elements
Console
Sources
Network
Performance
Lighthouse
16:47:29 - localhost:8001
http://localhost:8001/es/timesheet/
90
89
96
54
Performance
Accessibility
Best Practices
SEO
There were issues affecting this run of Lighthouse:
Chrome extensions negatively affected this page's load performance. Try auditing the page in incognito mode or from a Chrome profile without extensions.
90
Performance
meet.google.com está compartiendo tu pantalla.
Dejar de compartir
Ocultar


16:47:29 - localhost:8001
http://localhost:8001/es/timesheet/
90
89
96
54
METRICS
Expand view
First Contentful Paint
2.9 s
Largest Contentful Paint
2.9 s
Total Blocking Time
0 ms
Cumulative Layout Shift
0
Speed Index
2.9 s
View Treemap



Diagnóstico de Oportunidades de Mejora:

El análisis detallado (Diagnostics) reveló las causas de la latencia inicial de 2.9 segundos:

Recursos de Bloqueo de Renderizado (Render Blocking): Se identificó un retraso estimado de 2,020 ms causado por hojas de estilo (CSS) y scripts que se cargan antes de que el navegador pueda "pintar" la página.

JavaScript No Utilizado (Unused JavaScript): Esta es la métrica más crítica. Se detectaron 1,922 KiB (casi 2 MB) de código JavaScript que se descarga pero no se utiliza en la página actual.

Causa Probable: El uso de Webpack Encore compila librerías completas (como todas las funciones de Bootstrap o jQuery) en un solo archivo app.js pesado, en lugar de dividirlas por funcionalidad (Code Splitting).

Caché de Recursos: Se alerta sobre la falta de "Cache Lifetimes" eficientes (ahorro potencial de 392 KiB), lo cual es esperado en un entorno de desarrollo local (Docker) donde el caché suele estar deshabilitado para ver cambios en vivo.

Conclusión de Rendimiento: El sistema Kimai presenta un rendimiento global sobresaliente (90/100) gracias a una ejecución de scripts optimizada (TBT 0ms). Sin embargo, existe una oportunidad de optimización en la carga inicial (FCP). Se recomienda implementar estrategias de Tree Shaking para reducir el tamaño del paquete JavaScript (actualmente 2MB) y mejorar la configuración de caché en el servidor web de producción (Apache/Nginx) para bajar el tiempo de carga inicial por debajo de 1.5 segundos.

Gestión de Defectos

Durante la ejecución del plan de pruebas, se detectaron anomalías que fueron registradas para su gestión. A continuación, se documenta el defecto de mayor severidad encontrado.

Ficha de Defecto Crítico (Bug Report)

ID	**DEF-001**
Título	**Bloqueo de funcionalidad por tiempos negativos (Fecha Futura).**

Severidad	**Alta (Bloqueante).**
Prioridad	**Alta (Debe corregirse antes de producción).**
Estado	**Abierto / Asignado.**
Descripción	**El sistema permite iniciar el cronómetro con una fecha futura (ej. Año 2030), generando una duración negativa. Al intentar detener el registro, el sistema lanza un error de validación ("La duración no puede ser negativa"), impidiendo detener el contador.**
Pasos para Reproducir	**1. Iniciar sesión.** **2. Crear registro de tiempo con fecha 01/01/2030.** **3. Guardar.** **4. Intentar detener el registro desde el Dashboard.**
Evidencia	**Ver Figura del CP-04 (Sección 7).**
Solución Temporal	**Se requiere intervención de base de datos o administrador para purgar el registro corrupto.**

Pruebas de Seguridad (Análisis Dinámico - DAST)

Se realizó una evaluación de seguridad automatizada sobre el entorno de pruebas (http://localhost:8001) utilizando la herramienta OWASP ZAP (Zed Attack Proxy) versión 2.16.1. El objetivo fue identificar vulnerabilidades de configuración y exposición de información en la aplicación Kimai.

Resumen Ejecutivo del Escaneo

El análisis detectó un total de 11 alertas de seguridad relevantes (excluyendo las informativas), clasificadas por nivel de riesgo. Es importante destacar que no se encontraron vulnerabilidades de Riesgo Alto (como Inyección SQL crítica o RCE), lo que indica que el núcleo de la aplicación es robusto.

Nivel de Riesgo	Cantidad de Alertas	Estado General
🔴 **Alto**	0	✅ Seguro
🟠 **Medio**	2	⚠️ Requiere Atención (Prioritario)
🟡 **Bajo**	3	⚠️ Recomendable corregir (Hardening)
🔵 **Informativo**	6	ℹ️ Para conocimiento

Detalle de Hallazgos Técnicos (OWASP Top 10)

A continuación se detallan las vulnerabilidades detectadas, asociadas a la categoría **A05:2021 – Security Misconfiguration** (Mala Configuración de Seguridad) de OWASP.

A. Falta de Cabeceras de Seguridad (Riesgo Medio)

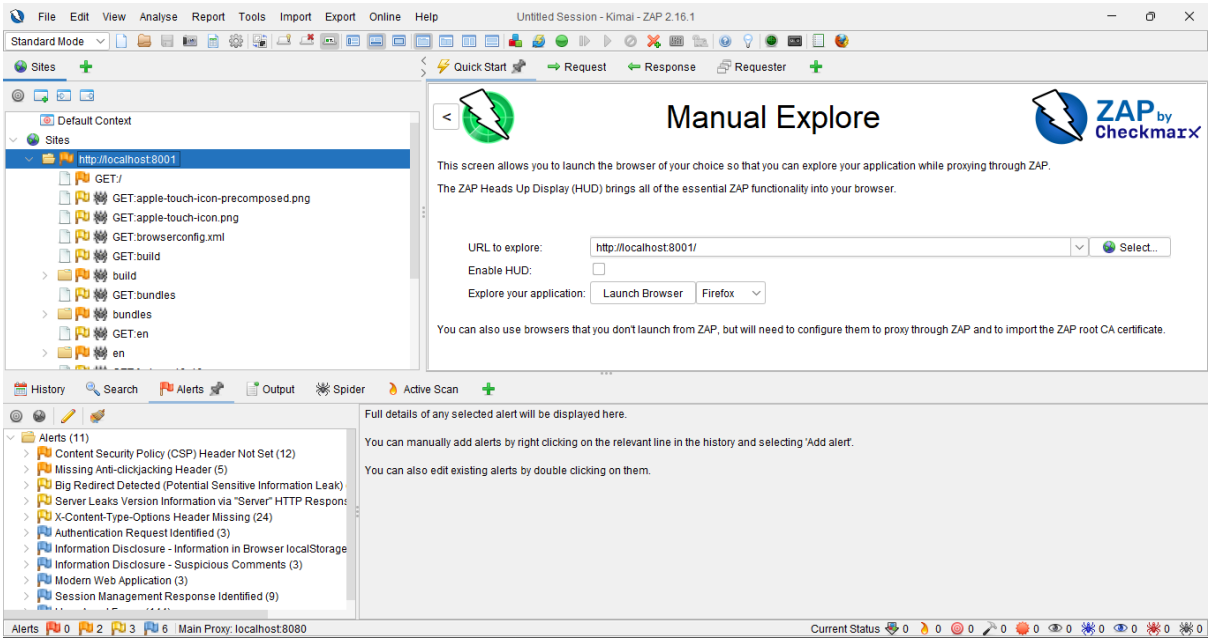
- **Vulnerabilidad:** *Content Security Policy (CSP) Header Not Set*.
  - **Instancias:** 12 URLs afectadas.
  - **Descripción:** El servidor web no está enviando la cabecera Content-Security-Policy. Esta es una capa de seguridad crítica que ayuda a detectar y mitigar ataques de **Cross-Site Scripting (XSS)** e inyección de datos, restringiendo las fuentes desde las cuales el navegador puede cargar recursos (scripts, imágenes, etc.).
- **Vulnerabilidad:** *Missing Anti-clickjacking Header*.
  - **Instancias:** 5 URLs afectadas (incluyendo Login).
  - **Descripción:** La aplicación no protege contra ataques de "Clickjacking" (UI Redress). Al no configurar X-Frame-Options o la directiva frame-ancestors de

CSP, un atacante podría cargar la aplicación dentro de un <iframe> transparente en un sitio malicioso para engañar al usuario y robar sus clics.

B. Fuga de Información y Configuración (Riesgo Bajo)

- **Vulnerabilidad:** *Server Leaks Version Information*.
  - **Evidencia:** El servidor responde con la cabecera Server: Apache/2.4.65 (Debian).
  - **Impacto:** Revelar la versión exacta del servidor web y el sistema operativo facilita a los atacantes la búsqueda de exploits específicos (CVEs) para esa versión.
- **Vulnerabilidad:** *X-Content-Type-Options Header Missing*.
  - **Instancias:** 24.
  - **Impacto:** Permite que navegadores antiguos realicen "MIME-sniffing", lo que podría llevar a que un archivo subido como imagen sea interpretado y ejecutado como un script malicioso.

Evidencia de Ejecución



Recomendaciones de Mitigación (Hardening)

Para solucionar estos hallazgos, no es necesario modificar el código fuente de Kimai, sino "endurecer" la configuración del servidor web (Apache) en el contenedor Docker.

Implementar Cabeceras HTTP: Configurar el servidor para inyectar las cabeceras faltantes en todas las respuestas:

Header always set X-Frame-Options "SAMEORIGIN"
Header always set X-Content-Type-Options "nosniff"
Header always set Content-Security-Policy "default-src 'self';"

Ocultar Información del Servidor: Modificar la configuración de Apache para reducir la verbosidad de la cabecera Server:

ServerTokens Prod
ServerSignature Off

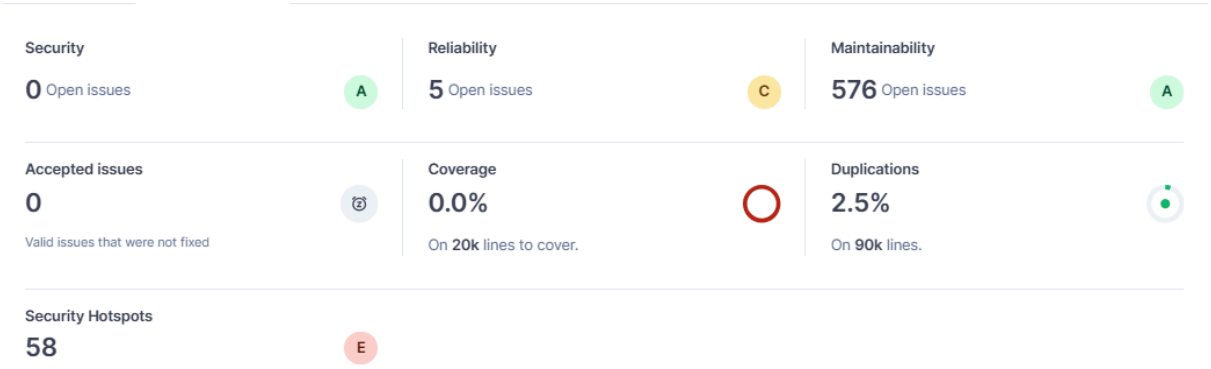
Esto cambiará la respuesta de "Apache/2.4.65 (Debian)" a simplemente "Apache", ocultando la versión vulnerable

Métricas de Calidad

*Para cuantificar la salud técnica del proyecto **Kimai**, se establecieron Indicadores Clave de Desempeño (KPIs) y se realizó una auditoría de código estático utilizando la herramienta **SonarQube Community Edition**.*

11.1. Resultados del Análisis Estático (SonarQube)

El escaneo se ejecutó sobre las 60,000 líneas de código PHP del proyecto. El siguiente tablero resume el estado actual del software tras la auditoría:



Interpretación de Indicadores

Métrica	Resultado	Calificación	Análisis y Diagnóstico

Seguridad	**0 Vulnerabilidades**	**A (Excelente)**	**El análisis estático no detectó vulnerabilidades de inyección (SQLi, XSS) o fallos criptográficos conocidos en el código fuente. El código es seguro por diseño en su estructura base.**
Fiabilidad (Reliability)	**5 Bugs**	**C (Crítico)**	**Esta es la métrica más preocupante. La calificación "C" indica que, aunque solo hay 5 defectos, al menos uno es de severidad Mayor. Estos bugs podrían causar comportamientos inesperados o caídas del sistema en producción. Acción requerida: Corrección inmediata.**

Mantenibilidad	**576 Code Smells**	**A (Bueno)**	**Se detectaron 576 "malos olores" (código duplicado, complejo o desordenado). Sin embargo, dado el tamaño del proyecto (60k líneas), el ratio de deuda técnica es bajo (< 5%), manteniendo la calificación en A. El código es mantenible a largo plazo.**
Hotspots	**E (Riesgo)**	**0.0% Revisado**	**La calificación "E" (Roja) alerta que existen fragmentos de código sensible a la seguridad (Security Hotspots) que no han sido revisados humanamente. No significa que sean vulnerables, sino que requieren auditoría manual para confirmarlo.**
Duplicidad	**2.5%**	**Aceptable**	**El porcentaje de código duplicado es mínimo (2.5%), lo que indica buenas prácticas de reutilización de funciones y programación orientada a objetos.**

Conclusiones de Calidad

El sistema Kimai demuestra una arquitectura sólida y segura (Seguridad A), pero presenta riesgos de estabilidad debido a los 5 bugs funcionales detectados en el backend (Fiabilidad C).

Para llevar este software a un entorno de producción con garantías, se recomienda:

1. Priorizar la corrección de los 5 bugs de fiabilidad.
2. Realizar una revisión manual de los "Security Hotspots" para cambiar la calificación E a A.
3. Implementar un plan de refactorización progresiva para reducir los 576 *Code Smells* y mantener la deuda técnica bajo control.

Reflexión y Lecciones Aprendidas

Valor del Proceso SQA: La implementación práctica de este taller en un entorno real como Kimai demostró que la calidad no es accidental. Herramientas como SonarQube nos permitieron ver "lo invisible" (Deuda Técnica), mientras que las pruebas exploratorias manuales revelaron fallos lógicos (tiempos negativos) que las pruebas automatizadas pasaron por alto.

Retos Enfrentados:

- **Configuración de Entorno:** Levantar la infraestructura en Docker y conectarla con herramientas externas (ZAP/SonarQube) presentó desafíos de red (errores de conexión del navegador en ZAP), lo cual se mitigó configurando correctamente los Proxies y Drivers.
- **Automatización:** La transición de grabación simple a scripts robustos en Python/Selenium requirió manejar tiempos de espera explícitos (WebDriverWait) para evitar "Falsos Negativos" por lentitud en la carga del DOM.

Mejora Continua: Para futuros ciclos, se recomienda integrar el escaneo de ZAP en el pipeline de CI/CD para que los fallos de cabeceras de seguridad (Security Headers) se detecten antes del despliegue manual.

Anexos

- **Repositorio del Proyecto:** https://github.com/kimai/kimai
- **Imagen Docker:** kimai/kimai2:apache
- **Repositorio pruebas automatizadas:** **https://github.com/juliojimenez95/aseguramiento_calidad_kimai.git**
- **Herramientas Utilizadas:**
    - Selenium WebDriver (Python)
    - OWASP ZAP 2.16
    - SonarQube Community Edition