Reporte de Pruebas de Aceptación

Sistema de Control de Asistencia GeoFace

Fecha de verificación: 10/11/2025

Requisitos Funcionales

14 / 14

Cumple

Requisitos No Funcionales

5 / 5

Cumple

Resumen Ejecutivo

Se realizaron pruebas de aceptación sobre 14 requisitos funcionales y 5 requisitos no funcionales del sistema GeoFace. El resultado global indica que todas las validaciones planificadas fueron aprobadas, asegurando el cumplimiento de las capacidades clave solicitadas por el cliente.

Requisitos Funcionales

RF-001

Autenticar usuario en el sistema

Escenario de Prueba:
Administrador inicia sesión con credenciales válidas (usuario@admin.com, password) y accede al dashboard correspondiente según su rol (ADMIN/EMPLEADO).

Resultado Obtenido:
La sesión se establece en menos de 2 segundos, el estado global refleja su rol, y si el usuario está inactivo, la sesión se cierra automáticamente.

Archivos de Implementación:
lib/controllers/auth_controller.dart
lib/services/auth_service.dart
lib/models/usuario.dart

Evidencia de Código:
auth_controller.dart:102-131
- Método login() valida credenciales
- _onAuthStateChanged() detecta cambios
- _fetchUserData() carga rol del usuario
- Validación de usuario activo (línea 74-81)

Pruebas Unitarias:
test/auth_controller_test.dart (5 tests) - Validación credenciales, roles, usuarios inactivos

Evidencias Visuales/Logs:
Video Login_Admin.mp4 / Logs auth_controller (2025-11-07) / Captura Dashboard según rol

Responsable: QA - Jorge Briceño

APROBADO

RF-002

Gestionar sedes con perímetros

Escenario de Prueba:
Administrador crea una nueva sede con nombre, dirección, coordenadas GPS (latitud, longitud) y radio permitido (metros). La sede se visualiza en la lista general y puede ser editada o eliminada.

Resultado Obtenido:
La sede queda disponible para asignaciones, se persisten todos los campos obligatorios en Firestore (colección sedes), y el radio permitido se usa para validar geocercas.

Archivos de Implementación:
lib/controllers/sede_controller.dart
lib/services/sede_service.dart
lib/models/sede.dart

Evidencia de Código:
sede_controller.dart:64-99
- addSede() crea sede con UUID
- updateSede() actualiza datos
- deleteSede() elimina sede
- Persistencia en Firestore (sede_service.dart:56-57)

Pruebas Unitarias:
test/sede_controller_test.dart (6 tests) - CRUD completo, validación coordenadas y radio

Evidencias Visuales/Logs:
Captura RF002_sede_creada.png / Registro en colección sedes / Mapa con marcador de sede

Responsable: QA - Brayar Lopez

APROBADO

RF-003

Gestionar empleados

Escenario de Prueba:
Se registra un empleado con nombre, apellidos, DNI, celular, correo, cargo y sede asignada. El sistema valida que el DNI y correo sean únicos antes de crear el registro.

Resultado Obtenido:
El empleado aparece activo en la tabla, sin duplicados, con datos consistentes y asignado a la sede correcta. Las validaciones previenen duplicados de DNI y correo.

Archivos de Implementación:
lib/controllers/empleado_controller.dart
lib/services/empleado_service.dart
lib/models/empleado.dart

Evidencia de Código:
empleado_controller.dart:116-135
- validarDatosUnicos() verifica DNI y correo
- addEmpleado() crea empleado con validación
- updateEmpleado() actualiza con validación
- Asignación a sede (campo sedeId)

Pruebas Unitarias:
test/empleado_controller_test.dart (7 tests) - CRUD, validación unicidad, filtrado por sede

Evidencias Visuales/Logs:
Checklist RF003_empleado.xlsx / Captura formulario con validaciones / Error al duplicar DNI

Responsable: QA - Jorge Briceño

APROBADO

RF-004

Registrar datos biométricos

Escenario de Prueba:
El sistema captura tres imágenes faciales mediante la cámara frontal, las sube a Firebase Storage y almacena las URLs en Firestore. El empleado queda marcado con hayDatosBiometricos=true.

Resultado Obtenido:
Se generan URLs válidas en Firebase Storage (ruta: biometricos/{empleadoId}/), se crea documento en colección biometricos, y se actualiza el flag hayDatosBiometricos en el empleado.

Archivos de Implementación:
lib/controllers/biometrico_controller.dart
lib/models/biometrico.dart

Evidencia de Código:
biometrico_controller.dart:139-187
- registerOrUpdateBiometricoWithMultipleFiles() valida 3 imágenes
- Upload a Storage (línea 158-161)
- Creación documento Firestore (línea 166-171)
- Actualización flag empleado (línea 174-177)

Pruebas Unitarias:
test/biometrico_controller_test.dart (6 tests) - Registro, recuperación, eliminación de datos biométricos

Evidencias Visuales/Logs:
Carpeta evidencias/biometrico/emp_001/ / Log Storage upload / Firestore documento biometricos

Responsable: QA - Brayar Lopez

APROBADO

RF-005

Configurar URLs de API

Escenario de Prueba:
Administrador ingresa una URL base HTTPS (ej: https://api.ejemplo.com) y el sistema deriva automáticamente los endpoints /identificar y /sync-database. La configuración se persiste en Firestore.

Resultado Obtenido:
Los endpoints son persistidos en Firestore (app_config/settings), validados (solo URLs HTTPS), y la sincronización queda habilitada para uso posterior.

Archivos de Implementación:
lib/controllers/api_config_controller.dart
lib/services/api_config_service.dart
lib/models/api_config.dart

Evidencia de Código:
api_config_controller.dart:58-85
- saveApiConfigFromBaseUrl() valida URL
- Construcción endpoints (línea 71-72)
- Persistencia Firestore (api_config_service.dart:39-48)
- Validación HTTPS implícita

Pruebas Unitarias:
test/api_config_controller_test.dart (7 tests) - Guardar, recuperar, actualizar configuración API

Evidencias Visuales/Logs:
Captura RF005_config.png / Documento app_config/settings en Firestore / Validación URL HTTPS

Responsable: QA - Jorge Briceño

APROBADO

RF-006

Marcar asistencia con reconocimiento facial

Escenario de Prueba:
Empleado con datos biométricos válidos marca entrada dentro de la geocerca (validación GPS), usando hora de red NTP. Luego registra salida con validación de ubicación. Se capturan imágenes faciales en ambos casos.

Resultado Obtenido:
Se crea el registro de asistencia con coordenadas GPS, hora de red (NTP), URL de captura facial, y validación de geocerca. La asistencia se persiste en Firestore (colección asistencias).

Archivos de Implementación:
lib/controllers/asistencia_controller.dart
lib/services/asistencia_service.dart
lib/services/time_service.dart
lib/services/location_service.dart
lib/utils/location_helper.dart

Evidencia de Código:
asistencia_controller.dart:93-150
- registrarEntrada() valida GPS, NTP, geocerca
- TimeService.getCurrentNetworkTime() (línea 104)
- LocationHelper.calcularDistancia() (línea 120-122)
- Validación radio permitido (línea 123-125)

Pruebas Unitarias:
test/asistencia_controller_test.dart (7 tests) - Registro entrada/salida, validación GPS, tiempo trabajado

Evidencias Visuales/Logs:
Video RF006_asistencia.mov / Documento firestore/asistencias / Logs validación GPS y NTP

Responsable: QA - Brayar Lopez

APROBADO

RF-007

Visualizar detalle de asistencia diaria

Escenario de Prueba:
Empleado autenticado revisa su historial de asistencias del día actual. Puede distinguir asistencias completas (con entrada y salida) e incompletas (solo entrada), ver horas, coordenadas y tiempo trabajado.

Resultado Obtenido:
El detalle presenta horas de entrada/salida, coordenadas GPS, estado de registro (completo/incompleto), y tiempo trabajado calculado. Los registros se ordenan por fecha descendente.

Archivos de Implementación:
lib/controllers/asistencia_controller.dart
lib/services/asistencia_service.dart
lib/models/asistencia.dart

Evidencia de Código:
asistencia_controller.dart:214-227
- getAsistenciasByEmpleado() recupera historial
- asistencia_service.dart:41-56 consulta Firestore
- Asistencia.registroCompleto (modelo línea 77)
- Asistencia.tiempoTrabajado (modelo línea 81-83)

Pruebas Unitarias:
test/asistencia_detalle_test.dart (5 tests) - Recuperación historial, formato, ordenamiento, filtrado por fechas

Evidencias Visuales/Logs:
Capturas Aplicación móvil - pantalla historial / Vista detalle con coordenadas y horas

Responsable: QA - Jorge Briceño

APROBADO

RF-008

Dashboard de monitoreo (Empleado)

Escenario de Prueba:
Empleado autenticado visualiza su estado actual (debe marcar entrada, debe marcar salida, jornada completa) y accede a atajos para marcar asistencia y ver historial. El dashboard se actualiza automáticamente.

Resultado Obtenido:
La interfaz cambia dinámicamente según el estado de asistencia del día, muestra botones contextuales (Entrada/Salida), y mantiene consistencia tras refrescar. Se muestran estadísticas del día.

Archivos de Implementación:
lib/controllers/asistencia_controller.dart
lib/services/asistencia_service.dart
lib/views/empleado/marcar_asistencia_page.dart

Evidencia de Código:
asistencia_controller.dart:66-90
- checkEmpleadoAsistenciaStatus() determina estado
- AsistenciaStatus enum (entrada/salida/completa)
- getAsistenciasDeHoy() (línea 229-243)
- Refresh automático con notifyListeners()

Pruebas Unitarias:
test/dashboard_empleado_test.dart (5 tests) - Obtención datos diarios, cálculo estadísticas, formato visualización

Evidencias Visuales/Logs:
GIF RF008_dashboard.gif / Capturas estados del dashboard / Logs Provider notifyListeners()

Responsable: QA - Brayar Lopez

APROBADO

RF-009

Generar reportes detallados

Escenario de Prueba:
Administrador selecciona rango de fechas (mes) y sede (opcional), y genera reporte con asistencias, ausencias, tardanzas (entrada después de 9:00) y porcentaje de asistencia. El reporte se puede exportar a PDF.

Resultado Obtenido:
El resumen muestra totales consistentes (asistencias, ausencias, tardanzas, porcentaje), agrupa datos por día, identifica empleados ausentes, y permite exportar a PDF con formato profesional.

☒ **Archivos de Implementación:**
lib/controllers/reporte_controller.dart
lib/services/asistencia_service.dart
lib/utils/pdf_report_generator.dart

☒ **Evidencia de Código:**
reporte_controller.dart:77-182
- generarReporteDetallado() calcula estadísticas
- Cálculo ausencias por día (línea 114-130)
- Cálculo tardanzas (línea 134-141)
- Exportación PDF (línea 185-201)

☒ **Pruebas Unitarias:**
test/reporte_controller_test.dart (6 tests) - Cálculo asistencias, ausencias, tardanzas, filtrado por sede/mes

☒ **Evidencias Visuales/Logs:**
Reporte generado: reportes/reporte_agosto.pdf / Captura pantalla reportes con filtros / PDF exportado

Responsable: QA - Jorge Briceño

APROBADO

RF-010

Cambiar contraseña de usuario

Escenario de Prueba:
Usuario autenticado (admin o empleado) ingresa contraseña actual y nueva contraseña (mínimo 6 caracteres). El sistema valida la contraseña actual mediante reautenticación y actualiza la nueva contraseña.

Resultado Obtenido:
La contraseña se actualiza en Firebase Auth, se valida que la contraseña actual sea correcta, se aplican políticas de seguridad (mínimo 6 caracteres), y se notifica al usuario del cambio exitoso.

Archivos de Implementación:
lib/controllers/auth_controller.dart
lib/utils/validators.dart

Evidencia de Código:
auth_controller.dart:140-173
- changePassword() valida contraseña actual
- Reautenticación (línea 152-156)
- updatePassword() (línea 157)
- validators.dart:42-52 valida longitud mínima

Pruebas Unitarias:
test/cambiar_contrasena_test.dart (6 tests) - Validación políticas, contraseña actual, longitud mínima

Evidencias Visuales/Logs:
Registro firebaseAuth.changePassword / SnackBar confirmación / Validación contraseña actual incorrecta

Responsable: QA - Brayar Lopez

APROBADO

RF-011

Exportar reportes a PDF

Escenario de Prueba:
Administrador genera un reporte de asistencias y lo exporta a PDF. El PDF incluye resumen estadístico, tabla detallada por día, identificación de tardanzas y ausencias, y formato corporativo con logos.

Resultado Obtenido:
Se genera PDF descargable con todas las métricas (asistencias, ausencias, tardanzas, porcentajes), tabla detallada por día, formato profesional, y capacidad de compartir/imprimir.

Archivos de Implementación:
lib/utils/pdf_report_generator.dart
lib/controllers/reporte_controller.dart

Evidencia de Código:
pdf_report_generator.dart:101-147
- generateAndSharePdf() crea documento
- _buildSummaryTable() genera resumen (línea 194-212)
- _buildDetails() genera tabla diaria (línea 224-257)
- Formato profesional con Printing.layoutPdf()

Pruebas Unitarias:
test/pdf_report_generator_test.dart (5 tests) - Estructura datos, formato tablas, resumen estadístico

Evidencias Visuales/Logs:
Archivo pdf/export_2025-11.pdf / Vista previa PDF / Compartir PDF funcional

Responsable: QA - Jorge Briceño

APROBADO

RF-012

Gestionar usuarios administradores

Escenario de Prueba:
Superadmin crea nuevos administradores (nombre, correo, contraseña), edita nombres de administradores existentes, y activa/desactiva administradores. Los usuarios inactivos no pueden iniciar sesión.

Resultado Obtenido:
El listado refleja cambios en tiempo real, se respetan los roles (tipoUsuario: ADMIN), los usuarios inactivos son rechazados en el login, y todas las operaciones se persisten en Firestore.

Archivos de Implementación:
lib/controllers/administrador_controller.dart
lib/services/administrador_service.dart
lib/models/usuario.dart

Evidencia de Código:
administrador_service.dart:53-87
- createAdminUser() crea en Firebase Auth y Firestore
- updateAdminUser() actualiza nombre (línea 90-96)
- toggleUserStatus() activa/desactiva (línea 99-105)
- auth_controller.dart:74-81 valida usuario activo

Pruebas Unitarias:
test/administrador_controller_test.dart (6 tests) - CRUD administradores, validación roles, estado activo/inactivo

Evidencias Visuales/Logs:
Captura RF012_admins.png / Registro usuarios colección / Prueba login usuario inactivo rechazado

Responsable: QA - Brayar Lopez

APROBADO

RF-013

Asignar credenciales a empleados

Escenario de Prueba:

Administrador selecciona un empleado sin usuario y genera credenciales automáticamente (correo: {dni}@geoface.com, contraseña: {dni}). El sistema marca debeCambiarContrasena=true y cierra la sesión del admin por seguridad.

Resultado Obtenido:

El empleado recibe credenciales en Firebase Auth, se crea documento en colección usuarios con tipo EMPLEADO, se actualiza flag tieneUsuario en el empleado, y la sesión del admin se cierra automáticamente.

Archivos de Implementación:

lib/controllers/empleado_controller.dart
lib/services/empleado_service.dart

Evidencia de Código:

empleado_controller.dart:258-305
- assignUserToEmpleado() genera credenciales
- Creación Firebase Auth (línea 267)
- Creación documento usuarios (línea 271-280)
- Actualización flag tieneUsuario (línea 283-284)
- Cierre sesión admin (línea 290)

Pruebas Unitarias:

test/asignar_credenciales_test.dart (5 tests) - Generación credenciales, creación usuario, flag debeCambiarContrasena

Evidencias Visuales/Logs:

Log creación FirebaseAuth / Documento usuarios en Firestore / Flag tieneUsuario actualizado / Sesión admin cerrada

Responsable: QA - Jorge Briceño

APROBADO

RF-014

Sincronizar datos faciales con API externa

Escenario de Prueba:
Administrador configura URL de API y ejecuta sincronización. El sistema envía petición POST al endpoint /sync-database y recibe respuesta del servidor remoto.

Resultado Obtenido:
La API responde 200 OK y se muestra notificación de éxito. Si hay error de conexión o la API falla, se muestra mensaje de error apropiado. El estado de sincronización se refleja en la UI.

Archivos de Implementación:
lib/controllers/api_config_controller.dart
lib/services/api_config_service.dart
lib/models/api_config.dart

Evidencia de Código:
api_config_controller.dart:88-110
- syncRemoteDatabase() ejecuta POST
- Validación URL configurada (línea 89-91)
- Petición HTTP POST (línea 97)
- Manejo respuesta 200/error (línea 99-105)

Pruebas Unitarias:
test/sincronizar_api_test.dart (9 tests) - Validación URL, petición POST, manejo respuestas, errores de conexión

Evidencias Visuales/Logs:
Captura Postman sync-database.png / Log consola HTTP 200 / Notificación éxito en UI / Prueba error conexión

Responsable: QA - Brayar Lopez

APROBADO

Requisitos No Funcionales

RNF-001

Rendimiento

Descripción:

El sistema debe responder en menos de 2 segundos para operaciones comunes (login, consulta, marcación).

Criterio de Aceptación:

Mediciones con cronómetro y logs automáticos durante 10 ejecuciones consecutivas en ambiente QA. Validación de tiempos de respuesta en operaciones críticas.

☒ **Evidencia de Código/Implementación:**

Implementación asíncrona:
- auth_controller.dart:102-131 (login async)
- asistencia_controller.dart:93-150 (registro async)
- Firebase queries optimizadas con índices
- Uso de FutureBuilder en vistas para carga no bloqueante

☒ **Medición y Evidencias Técnicas:**

Promedio login 1.4 s, registro asistencia 1.8 s, carga dashboard 1.2 s (Logs 2025-11-07). Todas las operaciones cumplen con el requisito de < 2 segundos.

Logs de rendimiento: logs/performance_2025-11-07.log
- Login: 1.2-1.6s (promedio 1.4s)
- Registro asistencia: 1.5-2.0s (promedio 1.8s)
- Carga dashboard: 1.0-1.4s (promedio 1.2s)
- Consultas Firestore: < 500ms promedio

APROBADO

RNF-002

Seguridad

Descripción:

Comunicación cifrada entre clientes y backend, autenticación con Firebase y roles segregados. Protección de datos sensibles y validación de permisos.

Criterio de Aceptación:

Validación de certificados HTTPS, revisión de reglas Firestore y pruebas de cuentas inactivas. Verificación de políticas de contraseñas y reautenticación.

Evidencia de Código/Implementación:

Implementación de seguridad:
- auth_controller.dart:74-81 (validación usuario activo)
- auth_controller.dart:140-173 (reautenticación para cambio contraseña)
- validators.dart:42-52 (validación longitud contraseña)
- api_config_controller.dart:58-85 (validación URL HTTPS)
- Firestore rules: usuarios solo acceden a sus datos

Medición y Evidencias Técnicas:

Todas las URLs usan https://, reglas Firestore restringen acceso por rol, usuario inactivo es rechazado, y contraseñas cumplen política mínima de 6 caracteres.

Revisión de seguridad: security_audit_2025-11-07.pdf
- Todas las URLs API usan HTTPS
- Firestore rules implementadas y probadas
- Usuario inactivo rechazado correctamente (prueba manual)
- Contraseñas mínimas de 6 caracteres validadas
- Reautenticación requerida para cambios sensibles

APROBADO

RNF-003

Disponibilidad

Descripción:

El servicio debe operar de 8:00 a 18:00 con mantenimiento planificado fuera del horario. Uptime alto y recuperación ante fallos.

Criterio de Aceptación:

Monitoreo de uptime con Firebase Status y registro manual durante 5 días hábiles. Validación de servicios Firebase (Auth, Firestore, Storage).

Evidencia de Código/Implementación:

Arquitectura de disponibilidad:
- Uso de Firebase (99.9% SLA)
- Manejo de errores: try-catch en operaciones críticas
- time_service.dart:36-54 (fallback a hora local si NTP falla)
- location_service.dart:37-67 (manejo errores GPS)
- Reintentos implícitos en Firebase SDK

Medición y Evidencias Técnicas:

Disponibilidad 99.2% en semana 44 (logs cloud functions) - sin caídas en horario laboral. Todos los servicios Firebase operativos durante período de prueba.

Reporte de disponibilidad: uptime_report_semana44.pdf
- Uptime: 99.2% (solo mantenimientos programados)
- Sin caídas en horario laboral (8:00-18:00)
- Firebase Status: todos los servicios operativos
- Tiempo de recuperación: < 1 minuto en casos de error temporal

APROBADO

RNF-004

Portabilidad

Descripción:

Compatibilidad con Android 8.0+ y diseño adaptable a pantallas; preparada para futuro soporte iOS. Responsive design y adaptación a diferentes tamaños de pantalla.

Criterio de Aceptación:

Pruebas en dispositivos/emuladores 5.5" y 6.7" Android; verificación de build iOS en Flutter. Validación de permisos y funcionalidades en diferentes versiones de Android.

☒ Evidencia de Código/Implementación:

Implementación multiplataforma:
- pubspec.yaml: minSdkVersion 26 (Android 8.0)
- Responsive design: uso de MediaQuery y LayoutBuilder
- Permisos multiplataforma: permission_handler
- Geolocator compatible Android/iOS
- Firebase multiplataforma (Android/iOS/Web)

☒ Medición y Evidencias Técnicas:

APK probado en Pixel 3a (Android 12) y Samsung A21 (Android 10); flutter build ios --no-tree-shake-icons exitoso. Diseño adaptable verificado en múltiples resoluciones.

Pruebas de portabilidad: portability_test_results.pdf
- Android 8.0 (API 26): ☒ Funcional
- Android 10 (Samsung A21): ☒ Funcional
- Android 12 (Pixel 3a): ☒ Funcional
- iOS build: ☒ Compilación exitosa
- Resoluciones: 5.5", 6.7" - ☒ Diseño adaptable
- Permisos GPS/Cámara: ☒ Funcionales en todas las versiones

APROBADO

RNF-005

Mantenibilidad

Descripción:

Código modular con buenas prácticas Flutter, documentación y separación de responsabilidades. Arquitectura clara y tests unitarios completos.

Criterio de Aceptación:

Revisión de arquitectura MVC/Provider, comentarios en controladores y cumplimiento de lint. Cobertura de tests unitarios y documentación de código.

☒ **Evidencia de Código/Implementación:**

Arquitectura y organización:
- Separación MVC: controllers/, services/, models/, views/
- Documentación en cabeceras (ej: time_service.dart:1-21)
- Uso de Provider para estado global
- Servicios reutilizables (time_service, location_service)
- Tests unitarios: test/*_test.dart (85+ tests)
- Validadores centralizados: utils/validators.dart

☒ **Medición y Evidencias Técnicas:**

Ejecución de flutter analyze sin errores; documentación en cabeceras y tests unitarios por módulo. Cobertura de tests: 14 requisitos funcionales cubiertos con 85+ tests unitarios.

Análisis de código: code_quality_report_2025-11-07.pdf
- flutter analyze: 0 errores, 0 warnings
- Cobertura tests: 85+ tests unitarios
- Documentación: 100% de servicios documentados
- Arquitectura: MVC/Provider implementada correctamente
- Separación responsabilidades: controllers, services, models
- Linting: Cumplimiento 100% de reglas flutter_lints

APROBADO