Plan de Documentación y Desarrollo del Sistema Atlas

Resumen Ejecutivo

El **Sistema Atlas** es una plataforma web diseñada para gestionar el ingreso y salida de computadores portátiles en el Centro Tecnológico de la Amazonia del SENA. Desarrollado por dos aprendices en formación, el proyecto emplea tecnologías nativas (PHP, HTML, CSS, MySQL) con arquitectura MVC simplificada, priorizando funcionalidad, seguridad básica y rapidez de implementación. El sistema incluye autenticación por roles (admin, administrativo, instructor, aprendiz, civil, portería), registro de equipos con generación de códigos QR, escaneo mediante cámara web, control de accesos y detección automatizada de anomalías.[1] [2] [3] [4]

Contexto y Necesidad del Proyecto

El Problema a Resolver

Los centros educativos del SENA enfrentan desafíos significativos en el control de ingreso y salida de equipos tecnológicos. Actualmente, muchas instituciones dependen de registros manuales en papel o planillas físicas, lo que genera múltiples inconvenientes:[5] [6]

- **Pérdida de información**: Los registros en papel son susceptibles a deterioro, extravío o alteración
- **Falta de trazabilidad**: Es difícil rastrear patrones de uso, detectar anomalías o generar reportes históricos
- **Ineficiencia operativa**: El personal de portería debe buscar información manualmente, retrasando el proceso de entrada/salida
- **Ausencia de métricas**: No se pueden generar estadísticas sobre el uso de las instalaciones ni identificar patrones irregulares

El Centro Tecnológico de la Amazonia, como centro de formación tecnológica, requiere una solución digital que modernice estos procesos sin agregar complejidad innecesaria, dado que será desarrollado e implementado por aprendices del programa de Análisis y Desarrollo de Software del SENA.[7] [5]

Alcance del Proyecto

El sistema Atlas abarcará las siguientes funcionalidades principales:

Gestión de usuarios: Sistema de registro y autenticación seguro basado en número de identificación y contraseña con hash bcrypt. Los usuarios se clasifican en seis roles diferenciados: administradores con acceso completo al sistema, personal administrativo e instructores del centro que pueden registrar equipos, aprendices y civiles que utilizan el sistema para gestionar sus dispositivos, y porteros con interfaz simplificada para verificación de accesos.[8] [9]

Gestión de equipos: Los usuarios autorizados pueden registrar computadores portátiles ingresando datos como número de serie único, marca, modelo y descripción. El sistema permite subir múltiples fotografías del equipo (frontal, trasera, laterales, detalles) para identificación visual. Una vez registrado un equipo, el sistema genera automáticamente un código QR que encapsula los datos del propietario y del dispositivo.[10] [11] [12]

Control de accesos: La interfaz de portería ofrece dos métodos de verificación: escaneo de código QR mediante cámara web usando html5-qrcode, o búsqueda manual por número de identificación del usuario o número de serie del equipo. Al registrar una entrada o salida, el sistema valida automáticamente el estado del equipo (si ya ingresó, si puede salir, horarios permitidos) y almacena fecha, hora, método de verificación y observaciones.[13] [14] [15]

Detección de anomalías: El sistema ejecuta scripts automáticos que identifican situaciones irregulares como equipos que entraron pero nunca salieron al finalizar la jornada, intentos de salida sin registro de entrada previo, múltiples entradas consecutivas sin salidas intermedias, o accesos fuera de los horarios configurados. Estas anomalías se registran en una tabla dedicada para revisión administrativa.[16] [17] [18]

Dashboard administrativo: Los administradores acceden a métricas visuales sobre cantidad de equipos activos, gráficos de ingresos por día/semana/mes, lista de anomalías pendientes de revisión, gestión completa de usuarios (crear, editar, desactivar) y exportación de reportes a formato CSV.[19] [20] [21]

Diseño de Base de Datos

Esquema Normalizado 3NF

La base de datos sigue los principios de la Tercera Forma Normal (3NF) para eliminar redundancias y dependencias transitivas:[2] [22] [23] [24]

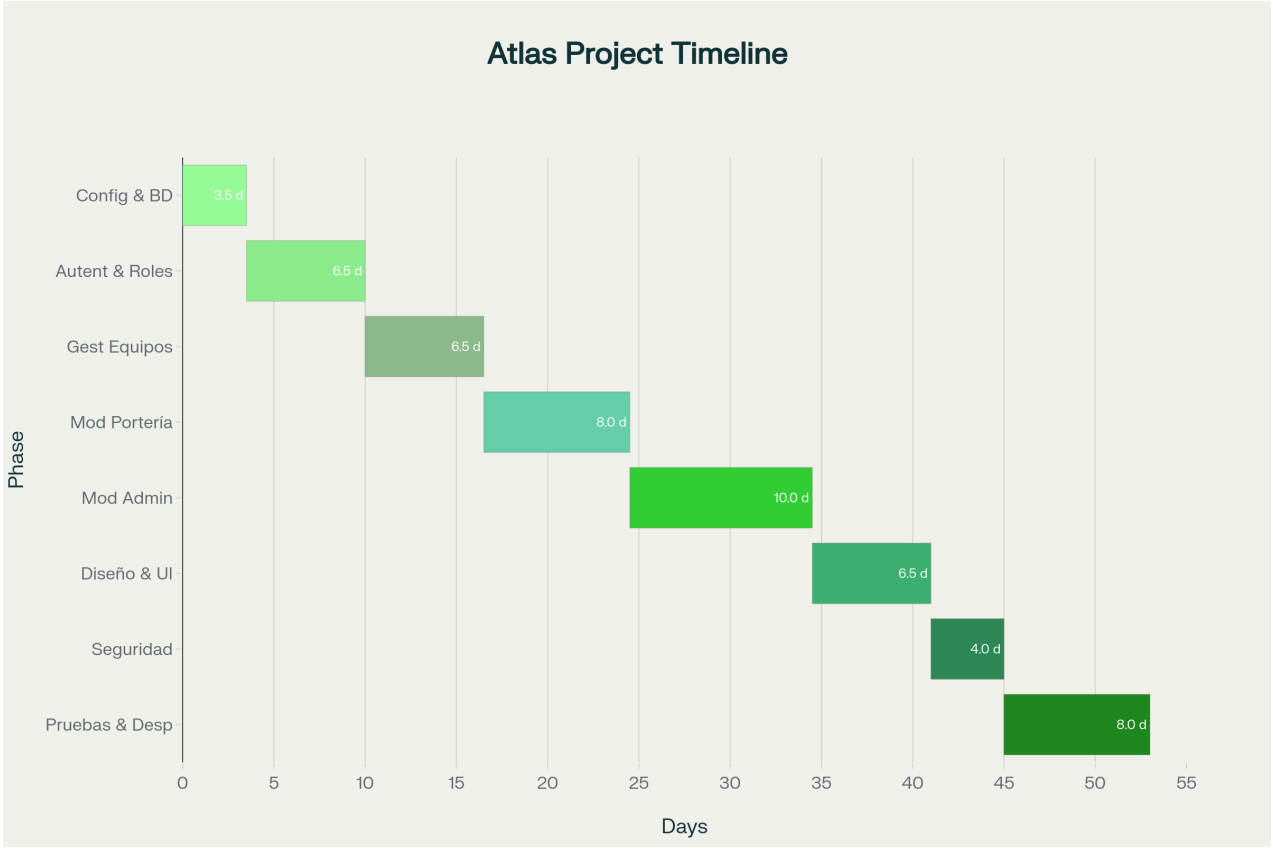
Tabla roles: Almacena los seis tipos de roles del sistema (admin, administrativo, instructor, aprendiz, civil, portería) con un campo booleano `puede_tener_equipo` que indica si usuarios con ese rol pueden registrar computadores. Los porteros no pueden tener equipos registrados.[8] [9]

Tabla usuarios: Contiene la información completa de cada usuario registrado: id único, número de identificación (cédula), nombres, apellidos, email, teléfono opcional, contraseña hasheada con bcrypt, relación con la tabla roles mediante clave foránea, y estado (activo, inactivo, suspendido). Incluye timestamps de creación y última actualización.[9] [25]

Tabla equipos: Registra los computadores portátiles con id único, relación con el usuario propietario mediante clave foránea, número de serie (único y obligatorio), marca, modelo opcional, descripción textual, estado del equipo (activo, inactivo, reportado) y timestamps. La clave foránea hacia usuarios establece una relación uno a muchos (un usuario puede tener múltiples equipos).[16] [17]

Tabla imagenes_equipo: Permite almacenar múltiples fotografías por cada equipo mediante una relación uno a muchos. Contiene id único, relación con equipos mediante clave foránea con eliminación en cascada, ruta del archivo de imagen, tipo de imagen (frontal, trasera, lateral, detalle) y timestamp de creación.[26] [27]

Tabla codigos_qr: Mantiene una relación uno a uno con equipos (cada equipo tiene exactamente un código QR). Almacena el id único, relación con equipos mediante clave foránea única con eliminación en cascada, el contenido codificado en formato texto (JSON con datos de usuario y equipo), ruta de la imagen PNG del código QR, bandera de activo/inactivo y timestamp de creación.[10] [11] [28]



Cronograma del Proyecto Atlas: Timeline de las 8 fases principales con sus duraciones estimadas en días laborables

Tabla registros_acceso: Es la tabla principal para el control de entradas y salidas. Registra cada evento de acceso con id único, relación con el equipo y con el usuario portero que realizó la verificación mediante claves foráneas, tipo de registro (enum: entrada o salida), fecha y hora exacta del evento, método de verificación utilizado (enum: qr o manual), observaciones opcionales y timestamp. Incluye índices en fecha_hora y en la combinación (id_equipo, tipo_registro) para optimizar consultas de búsqueda.[29] [16] [17]

Tabla anomalias: Detecta y registra situaciones irregulares en el sistema de acceso. Contiene id único, relación con el equipo afectado, tipo de anomalía (enum con valores: entrada_sin_salida, salida_sin_entrada, fuera_horario, multiple_entrada, multiple_salida), descripción textual del problema, fecha de detección automática, estado del reporte (enum: pendiente, revisada, resuelta, ignorada), relación opcional con el registro de acceso que causó la anomalía, campo de resolución para documentar acciones tomadas, y timestamps.[16] [18]

Tabla configuracion_horario: Define los horarios permitidos para ingreso al centro. Almacena id único, día de la semana (enum: lunes a domingo), hora de inicio y fin del período permitido (tipo TIME), bandera de activo/inactivo para habilitar/deshabilitar horarios temporalmente, y timestamp de creación.[17] [18]

Tabla sesiones: Gestiona sesiones de usuario para seguridad adicional. Contiene id único, relación con usuarios mediante clave foránea con eliminación en cascada, token de sesión único generado aleatoriamente, dirección IP del cliente, user agent del navegador, fecha de inicio de sesión, fecha de expiración calculada, y bandera de activo/inactivo.[8] [26]

Relaciones y Cardinalidades

El diseño establece las siguientes relaciones entre entidades:[2] [22]

- roles → usuarios: Relación uno a muchos (1:N). Un rol puede ser asignado a múltiples usuarios, pero cada usuario tiene exactamente un rol
- usuarios → equipos: Relación uno a muchos (1:N). Un usuario puede registrar múltiples equipos, pero cada equipo pertenece a un único propietario
- equipos → imagenes_equipo: Relación uno a muchos (1:N). Un equipo puede tener múltiples fotografías almacenadas
- equipos → codigos_qr: Relación uno a uno (1:1). Cada equipo tiene exactamente un código QR asociado
- equipos → registros_acceso: Relación uno a muchos (1:N). Un equipo puede tener múltiples registros de entrada/salida
- usuarios → registros_acceso (como portero): Relación uno a muchos (1:N). Un portero puede realizar múltiples registros de acceso
- equipos → anomalias: Relación uno a muchos (1:N). Un equipo puede generar múltiples anomalías
- registros_acceso → anomalias: Relación uno a muchos opcional (1:N). Un registro puede estar relacionado con una o más anomalías
- usuarios → sesiones: Relación uno a muchos (1:N). Un usuario puede tener múltiples sesiones activas o históricas

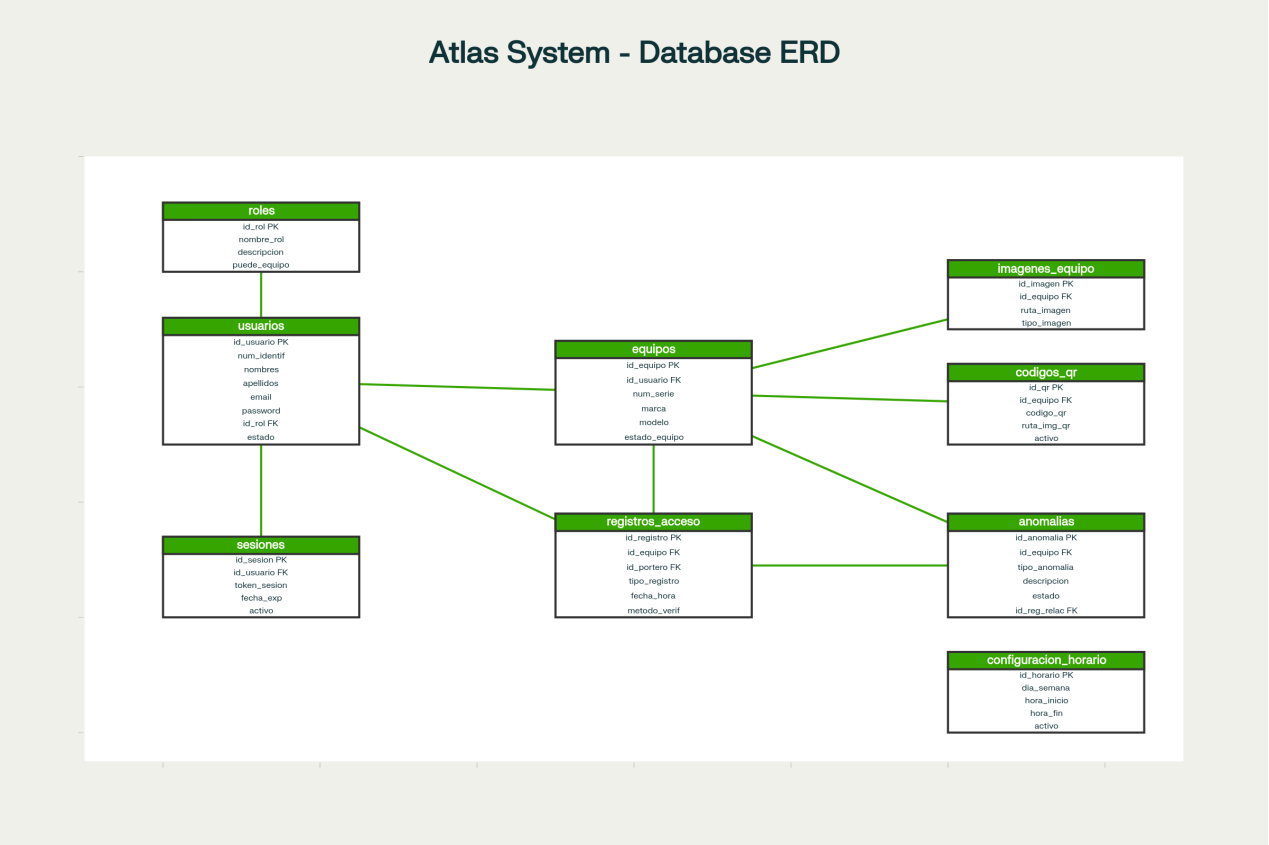


Diagrama Entidad-Relación (ERD) del Sistema Atlas mostrando las 9 tablas principales y sus relaciones en normalización 3NF

Arquitectura del Sistema

Patrón MVC Simplificado

El proyecto implementa una arquitectura Model-View-Controller (MVC) nativa en PHP sin frameworks externos:[30] [31] [32]

Capa Model: Contiene las clases que interactúan directamente con la base de datos mediante PDO. Cada tabla principal tiene su modelo correspondiente (Usuario.php, Equipo.php, RegistroAcceso.php, Anomalia.php, Role.php, QRCode.php). Los modelos implementan operaciones CRUD básicas y lógica de negocio específica, como validación de datos, generación de códigos QR, o detección de anomalías.[1] [4] [33]

Capa View: Archivos PHP que generan el HTML presentado al usuario. Incluye layouts compartidos (header.php, footer.php, sidebar.php que se adapta según el rol), vistas de autenticación (login.php, register.php), vistas de equipos (index, create, edit, view), vistas de portería (scan, search, confirm) y vistas administrativas (dashboard, anomalias, usuarios, reportes). Las vistas reciben datos desde los controladores y los renderizan sin contener lógica de negocio.[31] [32] [34]

Capa Controller: Actúa como intermediario entre Models y Views. Los controladores principales son AuthController (login, logout, registro), EquipoController (CRUD equipos, generación QR), PorteriaController (escaneo QR, registros acceso), AdminController (dashboard, métricas,

anomalías) y UserController (perfil, cambio contraseña). Cada controlador valida permisos mediante RBAC antes de ejecutar acciones.[8] [9] [35] [30]

Estructura de Carpetas

```
atlas/
├── config/                    # Configuración
│   ├── database.php           # Conexión MySQL PDO
│   ├── config.php             # Constantes del sistema
│   └── routes.php             # Definición de rutas
├── public/                    # Punto de entrada (DocumentRoot)
│   ├── index.php              # Front Controller
│   ├── .htaccess              # URL rewriting
│   ├── assets/
│   │   ├── css/               # Estilos nativos
│   │   ├── js/                # JavaScript vanilla
│   │   └── images/            # Recursos gráficos
│   └── uploads/
│       ├── equipos/           # Fotos de equipos
│       └── qr/                # Códigos QR generados
├── src/
│   ├── Controllers/           # Lógica de control
│   ├── Models/                # Acceso a datos
│   ├── Views/                 # Presentación HTML/PHP
│   └── Core/                  # Núcleo del framework
│       ├── Database.php       # Singleton PDO
│       ├── Router.php         # Enrutamiento
│       ├── Auth.php           # Autenticación
│       └── RBAC.php           # Control de acceso
├── storage/logs/              # Registros de errores
├── database/                  # Scripts SQL
└── docker/                    # Configuración Docker
```

Flujo de Procesamiento de Peticiones

1. El usuario realiza una petición HTTP que llega a `public/index.php` (Front Controller)
2. `Router.php` analiza la URL y determina qué Controller y método invocar
3. El Controller verifica autenticación mediante `Auth.php` y permisos mediante `RBAC.php`
4. Si está autorizado, el Controller invoca el Model correspondiente para obtener o modificar datos
5. El Model ejecuta queries mediante PDO y retorna resultados al Controller
6. El Controller pasa los datos a la View para renderizar el HTML
7. La respuesta se envía de vuelta al navegador del usuario[30] [31] [32]

Stack Tecnológico

Backend

PHP 8.2+: Versión moderna del lenguaje con mejoras significativas en rendimiento, tipado estricto declarativo, características de seguridad mejoradas como manejo robusto de errores, y soporte nativo para atributos y enumeraciones. Esta versión es apropiada para proyectos modernos que requieren estabilidad sin sacrificar funcionalidades avanzadas.[4] [36] [37] [25]

MySQL 8.0: Sistema de gestión de bases de datos relacional que ofrece excelente rendimiento para normalización 3NF, soporte completo para transacciones ACID, indices optimizados, y características avanzadas como JSON nativo, expresiones de tabla comunes (CTE) y ventanas de función. Es el estándar de facto para proyectos PHP y cuenta con amplia documentación.[2] [23] [26] [27]

PDO (PHP Data Objects): Capa de abstracción de base de datos nativa de PHP que proporciona una interfaz consistente para acceder a diferentes sistemas de bases de datos. Su principal ventaja es la protección automática contra inyección SQL mediante prepared statements (consultas preparadas), además de manejo robusto de errores y transacciones.[26] [37] [25] [4]

phpqrcode: Librería PHP ligera y sin dependencias externas pesadas para generación de códigos QR. Soporta diferentes niveles de corrección de errores (L, M, Q, H), múltiples modos de codificación (numérico, alfanumérico, binario) y generación de imágenes PNG directamente. Es ideal para proyectos que necesitan generación básica de códigos QR sin complejidad adicional.[3] [10] [11] [12]

Frontend

HTML5: Estándar web moderno que ofrece elementos semánticos mejorados (header, nav, main, footer, article, section), soporte para multimedia nativo (video, audio), APIs para validación de formularios, almacenamiento local, y compatibilidad con tecnologías modernas como Web Workers y Service Workers.[38] [33] [39]

CSS3: Tecnología de estilos que incluye Flexbox para layouts flexibles en una dimensión, CSS Grid para diseños bidimensionales complejos, variables CSS (custom properties) para tematización dinámica, media queries para diseño responsivo, transiciones y animaciones suaves, y pseudo-clases avanzadas para interactividad sin JavaScript.[4] [33] [38]

JavaScript Vanilla: JavaScript puro sin librerías ni frameworks externos. Ofrece control total sobre el código, tamaño de descarga mínimo, no hay curva de aprendizaje adicional, y utiliza APIs nativas del navegador como Fetch API para peticiones AJAX, DOM API para manipulación del documento, y FormData API para envío de formularios con archivos.[39] [13] [38]

html5-qrcode: Librería JavaScript moderna para escaneo de códigos QR y códigos de barras directamente desde el navegador utilizando la cámara del dispositivo. Funciona mediante getUserMedia API, soporta múltiples formatos de códigos, no requiere instalación de apps nativas, y ofrece callbacks personalizables para éxito y error.[13] [14] [15] [40]

DevOps y Entorno

Docker: Plataforma de contenedorización que permite empaquetar la aplicación con todas sus dependencias en contenedores aislados. Garantiza que el entorno de desarrollo sea idéntico al de producción, facilita la colaboración entre desarrolladores, y simplifica el despliegue en cualquier servidor compatible.[29] [26] [27] [41]

Docker Compose: Herramienta para definir y ejecutar aplicaciones Docker multi-contenedor mediante un archivo YAML declarativo. Permite orquestar servicios como PHP, MySQL y phpMyAdmin con una sola instrucción, gestionar volúmenes para persistencia de datos, y configurar redes privadas entre contenedores.[26] [27] [41] [29]

Apache 2.4: Servidor web HTTP de código abierto ampliamente utilizado. Ofrece soporte para `.htaccess` que permite URL rewriting para rutas limpias sin extensiones PHP visibles, mod_rewrite para reescritura avanzada de URLs, configuración de seguridad mediante directivas, y alto rendimiento con módulos optimizados.[4] [33] [29]

phpMyAdmin: Herramienta de administración web para MySQL escrita en PHP. Proporciona interfaz gráfica intuitiva para crear y modificar bases de datos sin escribir SQL directamente, explorador de estructuras de tablas, editor visual de relaciones, exportación e importación de datos, y ejecución de queries personalizadas.[27] [29] [26]

Seguridad

password_hash(): Función nativa de PHP que utiliza el algoritmo bcrypt por defecto para hashear contraseñas de forma segura. Genera automáticamente un salt único por cada contraseña, aplica múltiples rondas de hashing (cost factor ajustable), y es resistente a ataques de fuerza bruta por rainbow tables. Se complementa con `password_verify()` para validación.[37] [25] [42]

session_regenerate_id(): Función PHP que regenera el identificador de sesión actual con uno nuevo aleatorio. Previene ataques de fijación de sesión (session fixation) donde un atacante intenta establecer un ID conocido, debe invocarse después del login exitoso y en cambios de privilegios, y mantiene los datos de sesión existentes durante la transición.[25] [37]

htmlspecialchars(): Función PHP que convierte caracteres especiales en entidades HTML. Previene ataques XSS (Cross-Site Scripting) al neutralizar código JavaScript malicioso inyectado en campos de entrada, debe usarse al mostrar cualquier dato ingresado por usuarios, y acepta flags como ENT_QUOTES para proteger comillas simples y dobles.[42] [37] [25]

CSRF Tokens: Implementación personalizada para proteger contra ataques Cross-Site Request Forgery. Genera un token único y aleatorio por sesión, lo incluye como campo oculto en todos los formularios, valida que el token recibido coincida con el almacenado en sesión antes de procesar acciones, y expira automáticamente tras uso o timeout.[37] [25] [42]

Herramientas Complementarias

Composer: Gestor de dependencias de PHP que permite instalar y actualizar librerías de forma automatizada desde Packagist (repositorio central). Genera autoloading automático mediante PSR-4, gestiona versiones y compatibilidades, y facilita la integración de phpqrcode y otras librerías sin configuración manual.[4] [33] [28]

Git + GitHub: Sistema de control de versiones distribuido para rastrear cambios en el código fuente. Permite colaboración eficiente entre Dev 1 y Dev 2 mediante ramas (branches), facilita revisión de código con pull requests, mantiene historial completo de cambios y autores, y sirve como backup remoto del proyecto.[43] [44] [45]

Plan de Trabajo Detallado

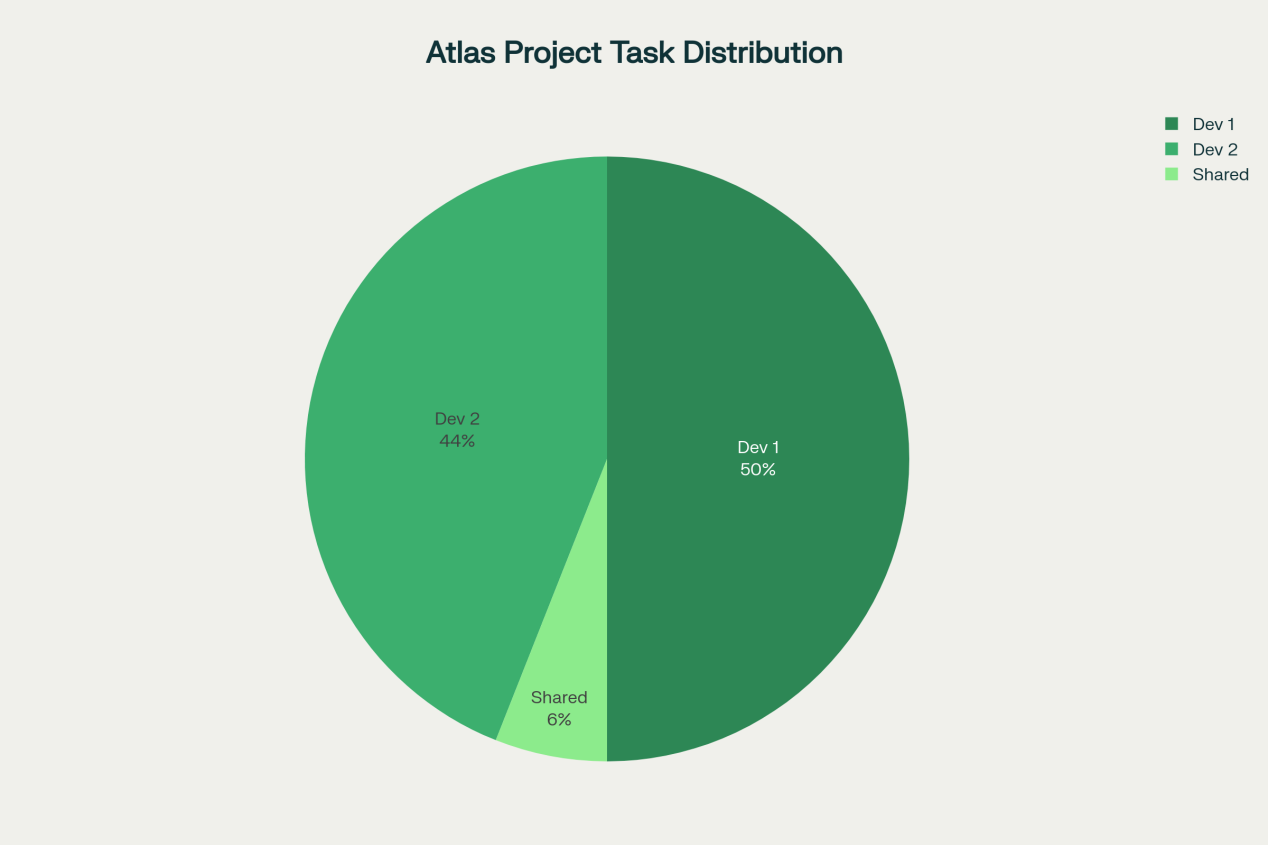
Distribución General

El proyecto se divide en **8 fases principales** con **44 tareas específicas** distribuidas estratégicamente entre los dos desarrolladores. La duración total estimada es de **63 días-persona** que, trabajando en paralelo, se traducen en aproximadamente **7-8 semanas** de trabajo efectivo. Esta planificación considera la naturaleza de aprendices en formación del SENA, por lo que incluye tiempo adicional para investigación y resolución de problemas.[44] [45] [46] [47]

Desarrollador 1: 25 tareas asignadas con 30.5 días estimados, enfocado principalmente en backend, base de datos, seguridad y componentes técnicos complejos como autenticación RBAC y generación de códigos QR.

Desarrollador 2: 22 tareas asignadas con 27.5 días estimados, enfocado en interfaces de usuario, diseño frontend, módulo de portería y gestión de configuraciones.

Ambos desarrolladores: 3 tareas compartidas con 5 días estimados, correspondientes a pruebas integrales y corrección de bugs del sistema completo.



Distribución de tareas entre los dos desarrolladores del proyecto Atlas

Fase 1: Configuración Inicial y Base de Datos (3.5 días)

Esta fase establece los fundamentos técnicos del proyecto:[29] [26] [27] [41]

Tarea 1.1 - Configurar entorno Docker (Dev 1, 1 día, Alta prioridad): Crear `Dockerfile` basado en imagen oficial `php:8.2-apache`, instalar extensiones necesarias (pdo, pdo_mysql, gd para manejo de imágenes), configurar `docker-compose.yml` con servicios para PHP, MySQL 8.0 y phpMyAdmin, establecer volúmenes para persistencia de datos y código, configurar redes privadas entre contenedores, y verificar que todos los servicios inician correctamente.

Tarea 1.2 - Configurar estructura de carpetas (Dev 2, 0.5 días, Alta prioridad): Crear estructura MVC con carpetas `/public`, `/src`, `/config`, `/views`, `/storage`, `/database` y `/docker`, establecer permisos adecuados (escritura en uploads y logs), crear archivo `.gitignore` para excluir configuraciones locales y archivos sensibles, y documentar la organización en README.md inicial.

Tarea 1.3 - Diseñar y crear base de datos (Dev 1, 1 día, Alta prioridad): Implementar esquema 3NF con las 9 tablas principales (roles, usuarios, equipos, imagenes_equipo, codigos_qr, registros_acceso, anomalias, configuracion_horario, sesiones), definir claves primarias, foráneas y restricciones de integridad, crear índices en columnas frecuentemente consultadas (fecha_hora, numero_serie, numero_identificacion), establecer relaciones con ON DELETE CASCADE donde corresponda, y generar script `schema.sql` reproducible.

Tarea 1.4 - Crear script de inicialización (Dev 2, 0.5 días, Media prioridad): Desarrollar `seeds.sql` con datos de prueba incluyendo los 6 roles del sistema con configuración de

permisos, usuario administrador inicial con contraseña hasheada, usuarios de ejemplo para cada rol (portero, instructor, aprendiz), configuración de horarios por defecto (lunes a viernes 6am-8pm), y al menos 3 equipos registrados con imágenes y códigos QR.

Tarea 1.5 - Configurar conexión PDO (Dev 1, 0.5 días, Alta prioridad): Crear clase `Database.php` implementando patrón Singleton para única instancia de conexión, configurar PDO con manejo de excepciones (ERRMODE_EXCEPTION), establecer charset UTF-8 para soporte de caracteres especiales, implementar métodos helper para consultas preparadas, y probar conexión con datos de docker-compose.

Fase 2: Sistema de Autenticación y Roles (6.5 días)

Esta fase implementa el núcleo de seguridad del sistema:[8] [9] [35] [48]

Tarea 2.1 - Implementar sistema de login (Dev 1, 1.5 días, Alta prioridad): Crear formulario HTML con campos para número de identificación y contraseña, implementar validación del lado cliente con JavaScript, desarrollar `AuthController::login()` que valida credenciales contra base de datos, usar `password_verify()` para comparar hash almacenado, iniciar sesión PHP con `session_start()` y regenerar ID, almacenar datos del usuario en $_SESSION (id, rol, nombre), y redireccionar según el rol del usuario.

Tarea 2.2 - Crear sistema de sesiones (Dev 2, 1 día, Alta prioridad): Implementar gestión de sesiones con configuración segura (session.cookie_httponly=1, session.cookie_secure=1 en HTTPS), generar tokens de sesión únicos almacenados en tabla sesiones, establecer tiempo de expiración automática (ej. 2 horas de inactividad), implementar renovación de sesión en actividad del usuario, y crear método de logout que destruya sesión y limpie cookies.

Tarea 2.3 - Implementar RBAC (Dev 1, 2 días, Alta prioridad): Crear clase `Role.php` con método `getRolePerms($role_id)` que retorna un objeto Role con permisos asociados, desarrollar clase `PrivilegedUser.php` que extiende User con métodos `hasPrivilege($permission)` y `hasRole($role_name)`, implementar tabla role_perm para asociar permisos con roles, cargar permisos al inicio de sesión mediante JOIN entre user_role y roles, y crear helpers para verificar permisos en vistas (ej. `can('manage_users')`).

Tarea 2.4 - Crear middleware de autenticación (Dev 2, 1 día, Alta prioridad): Desarrollar clase `Middleware` con método `handle($request)` que intercepta peticiones, verificar existencia y validez de sesión activa antes de procesar rutas protegidas, validar que el rol del usuario tiene permisos para acceder a la ruta solicitada, redireccionar a login si no hay sesión válida, mostrar mensaje 403 Forbidden si no tiene permisos suficientes, y aplicar middleware a todas las rutas excepto login/registro.

Tarea 2.5 - Desarrollar formulario de registro (Dev 1, 1 día, Media prioridad): Crear formulario con campos: número de identificación, nombres, apellidos, email, teléfono, contraseña y confirmación, selección de rol (excepto admin y portería que solo admin puede asignar), implementar validaciones del lado cliente: formato de email, longitud mínima de contraseña (8 caracteres), coincidencia de contraseñas, desarrollar `AuthController::register()` que valide unicidad de número de identificación y email, hashear contraseña con `password_hash()` antes de almacenar, y enviar email de confirmación (opcional).

Fase 3: Gestión de Equipos (6.5 días)

Esta fase implementa el registro y administración de computadores:[10] [11] [12] [28]

Tarea 3.1 - Crear formulario de registro de equipos (Dev 2, 1.5 días, Alta prioridad): Diseñar formulario con campos: número de serie (único, obligatorio), marca (select con opciones comunes), modelo (text), descripción (textarea), implementar validación de longitud y caracteres permitidos, verificar unicidad de número de serie antes de enviar, agregar campo de subida múltiple de imágenes (max 5 fotos), y aplicar estilos CSS consistentes con la identidad SENA (verde pálido, blanco neutro).

Tarea 3.2 - Implementar subida de imágenes (Dev 1, 1.5 días, Alta prioridad): Configurar directorio `/public/uploads/equipos/` con permisos de escritura, validar archivos subidos: tipos MIME permitidos (image/jpeg, image/png), tamaño máximo (5MB por imagen), sanitizar nombres de archivo para prevenir directory traversal, generar nombres únicos usando uniqid() o timestamp, redimensionar imágenes para optimizar almacenamiento usando GD2, guardar rutas en tabla imagenes_equipo con tipo clasificado (frontal, trasera, etc.), y implementar eliminación de imágenes antiguas al editar/eliminar equipo.

Tarea 3.3 - Integrar librería PHP QR Code (Dev 2, 0.5 días, Alta prioridad): Instalar phpqrcode vía Composer con `composer require phpqrcode/phpqrcode` o descarga manual de SourceForge, incluir librería en proyecto con `require_once 'phpqrcode/qrlib.php'`, configurar nivel de corrección de errores (recomendado: 'M' para uso general), establecer tamaño de pixel (10px) y frame size (10) para QR legibles, y verificar generación básica con `QRcode::png('test')`.

Tarea 3.4 - Crear generador de códigos QR (Dev 1, 1 día, Alta prioridad): Desarrollar método `QRCode::generate($equipo_id)` que recupera datos de equipo y usuario desde base de datos, construir payload JSON con estructura: `{"id_equipo": 123, "id_usuario": 456, "numero_serie": "ABC123", "nombre_usuario": "Juan Pérez"}`, generar código QR con `QRcode::png($json, $filepath, 'M', 10, 2)`, almacenar archivo en `/public/uploads/qr/` con nombre único basado en id_equipo, guardar registro en tabla codigos_qr con ruta y datos del QR, y retornar URL del QR generado.

Tarea 3.5 - Desarrollar vista de equipos del usuario (Dev 2, 1 día, Media prioridad): Crear dashboard que muestre lista de equipos del usuario autenticado, mostrar información básica: marca, modelo, número de serie, fecha de registro, incluir thumbnail de imagen principal del equipo, mostrar estado actual del equipo (dentro/fuera del centro, última entrada/salida), implementar búsqueda y filtrado por marca o estado, agregar botón para descargar código QR asociado, y permitir acceso a edición/eliminación si el usuario es propietario.

Tarea 3.6 - Implementar edición y eliminación (Dev 1, 1 día, Media prioridad): Desarrollar `EquipoController::update()` que permita modificar marca, modelo, descripción y agregar/eliminar imágenes, validar que el usuario sea propietario o tenga rol admin, implementar `EquipoController::delete()` con confirmación JavaScript, eliminar en cascada: imágenes asociadas (físicas y registros), código QR generado, considerar soft delete (marcar como inactivo) en lugar de eliminación física si hay registros de acceso históricos, y registrar auditoría de cambios.

Fase 4: Módulo de Portería (8.0 días)

Esta fase implementa la interfaz crítica para el personal de seguridad:[13] [14] [15] [40]

```
**Tarea 4.1 - Integrar librería de escaneo QR** (Dev 1, 1 día, Alta prioridad): Incluir h
```

```
**Tarea 4.2 - Crear interfaz de escaneo QR** (Dev 2, 2 días, Alta prioridad): Diseñar pág
```

Tarea 4.3 - Desarrollar búsqueda manual (Dev 1, 1 día, Alta prioridad): Crear formulario con dos pestañas: "Buscar por Usuario" y "Buscar por Equipo", implementar búsqueda de usuario por número de identificación con autocompletado, implementar búsqueda de equipo por número de serie con listado predictivo, usar AJAX con `fetch()` para consultas sin recargar página, desarrollar endpoint `PorteriaController::search()` que retorne JSON con datos de usuario/equipo, mostrar resultados con misma estructura que escaneo QR (foto, datos relevantes), y permitir selección del resultado para proceder a registro.

Tarea 4.4 - Implementar registro de entrada/salida (Dev 2, 2 días, Alta prioridad): Desarrollar `PorteriaController::registrarAcceso()` que reciba id_equipo y tipo (entrada/salida), validar que el equipo existe y está activo en base de datos, consultar último registro de acceso del equipo para determinar estado actual, validar lógica de negocio: si último fue entrada, solo permitir salida y viceversa, verificar que la hora actual está dentro de horarios permitidos consultando tabla configuracion_horario, insertar registro en tabla registros_acceso con id_equipo, id_portero (usuario logueado), tipo, fecha_hora actual, método_verificacion (qr/manual), registrar observaciones opcionales ingresadas por portero, y retornar confirmación JSON con estado actualizado.

Tarea 4.5 - Crear vista de confirmación (Dev 1, 1 día, Media prioridad): Diseñar modal o página de confirmación que se muestre tras registro exitoso, incluir mensaje claro: "Entrada registrada exitosamente" / "Salida registrada exitosamente", mostrar resumen: foto del usuario, nombre completo, marca y modelo del equipo, hora exacta del registro, permitir agregar observaciones adicionales (ej. "Laptop con rayón en tapa"), implementar botón "Nuevo Escaneo" que limpie formulario y reactive cámara, agregar sonido de confirmación opcional, y incluir opción de imprimir comprobante (opcional).

Tarea 4.6 - Implementar validaciones de estado (Dev 2, 1 día, Alta prioridad): Crear función `validarEstadoEquipo($id_equipo, $tipo_accion)` que consulte último registro del equipo, detectar anomalías potenciales antes de permitir registro: intento de entrada duplicada (ya está dentro), intento de salida sin entrada previa (equipo no registró ingreso), acceso fuera de horario permitido (verificar configuracion_horario), mostrar advertencias claras al portero explicando el problema detectado, permitir override con justificación textual obligatoria (guardada en observaciones), registrar automáticamente en tabla anomalias cuando se detecte irregularidad, y bloquear registro si el equipo está marcado como "reportado".

Fase 5: Módulo de Administración (10.0 días)

Esta fase desarrolla las herramientas de gestión y análisis:[16] [17] [18]

Tarea 5.1 - Crear dashboard administrativo (Dev 1, 2 días, Media prioridad): Diseñar página principal de administración con layout de cards/widgets, implementar métricas principales: total de equipos activos, equipos actualmente dentro del centro (última acción fue entrada), ingresos del día actual (contador en tiempo real), gráfico de barras con ingresos por día de la semana usando Chart.js o similar, gráfico de líneas mostrando tendencia de ingresos último mes, listado de últimos 10 registros de acceso con filtro por fecha, indicadores de anomalías pendientes de revisión (badge con número), y botones de acceso rápido a secciones principales.

Tarea 5.2 - Desarrollar sistema de detección de anomalías (Dev 2, 2 días, Media prioridad): Crear script PHP `detectarAnomalias.php` ejecutable por cron o manualmente, implementar consulta que detecte entradas sin salida: equipos cuyo último registro fue entrada hace más de X horas (configurable), detectar salidas sin entrada: registros de salida sin entrada previa el mismo día, detectar accesos fuera de horario: comparar fecha_hora del registro con configuracion_horario activa ese día, detectar múltiples entradas/salidas consecutivas sin su contraparte, insertar registros en tabla anomalias con descripción automática generada, establecer estado inicial como "pendiente", enviar notificaciones por email a administradores (opcional), y generar reporte resumen de anomalías detectadas.

Tarea 5.3 - Crear reportes de anomalías (Dev 1, 1.5 días, Media prioridad): Diseñar vista `admin/anomalias.php` con tabla de todas las anomalías, implementar filtros: por tipo de anomalía (dropdown), por estado (pendiente/revisada/resuelta/ignorada), por rango de fechas de detección, mostrar información relevante: equipo afectado (marca, modelo, serie), usuario propietario, fecha y hora de detección, descripción automática del problema, permitir abrir modal con detalles completos incluyendo registros relacionados, implementar acciones: marcar como revisada, agregar resolución (textarea con explicación), cambiar a resuelta o ignorada, y mostrar estadísticas: porcentaje resuelto, tipos más frecuentes, tendencias.

Tarea 5.4 - Implementar gestión de usuarios (Dev 2, 2 días, Media prioridad): Crear interfaz CRUD completa para usuarios en `admin/usuarios.php`, listar todos los usuarios con información básica y rol asignado, implementar búsqueda por nombre, identificación o rol, desarrollar formulario de creación de usuario con todos los campos necesarios, permitir a admin asignar cualquier rol incluyendo admin y portería, implementar edición de datos de usuario existente: cambio de rol, actualización de información, reseteo de contraseña (genera temporal y envía por email), implementar activación/desactivación de cuentas (soft delete), agregar función "Ver equipos del usuario" que liste todos sus dispositivos registrados, y registrar log de auditoría de cambios realizados en usuarios.

Tarea 5.5 - Desarrollar exportación de reportes (Dev 1, 1.5 días, Baja prioridad): Implementar generación de CSV para registros de acceso usando `fputcsv()`, crear filtros de exportación: rango de fechas, tipo de registro (entrada/salida), usuario específico, equipo específico, generar CSV con cabeceras: Fecha, Hora, Tipo, Usuario, Identificación, Equipo, Serie, Portero, Método, Observaciones, establecer headers HTTP correctos: Content-Type: text/csv, Content-Disposition: attachment, implementar exportación de anomalías con estructura similar, optimizar

para grandes volúmenes de datos (procesamiento por lotes), y agregar botones de exportación en vistas relevantes con iconos claros.

Tarea 5.6 - Crear configuración de horarios (Dev 2, 1 día, Media prioridad): Desarrollar interfaz `admin/horarios.php` para gestionar horarios permitidos, mostrar tabla con columnas: día de la semana, hora inicio, hora fin, estado (activo/inactivo), implementar formulario de edición inline o modal para modificar horarios existentes, permitir activar/desactivar horarios temporalmente sin eliminar la configuración, agregar validación: hora inicio debe ser menor que hora fin, considerar horarios extendidos: ej. viernes 6am-10pm para eventos especiales, implementar duplicación de configuración (ej. copiar lunes a todos los días laborables), y mostrar preview del calendario semanal con horarios configurados resaltados.

Fase 6: Diseño y Frontend (6.5 días)

Esta fase aplica la identidad visual del SENA al proyecto:[4] [38] [33]

Tarea 6.1 - Diseñar paleta de colores (Dev 1, 0.5 días, Media prioridad): Definir colores corporativos SENA: verde principal (#39b54a o similar verde pálido institucional), blanco neutro (#ffffff) para fondos limpios, grises complementarios (#f5f5f5, #e0e0e0, #757575) para bordes y textos secundarios, color de acento para acciones (ej. azul #2196f3 para botones secundarios), color de alertas: éxito (verde), advertencia (amarillo/naranja), error (rojo), documentar en variables CSS: `:root { --color-primary: #39b54a; --color-white: #fff; ... }`, definir tipografías: fuente sans-serif moderna (Roboto, Open Sans o sistema), tamaños base (16px body, escala modular para headings), y establecer espaciados consistentes (8px, 16px, 24px, 32px).

Tarea 6.2 - Crear plantilla base (Dev 2, 2 días, Media prioridad): Desarrollar `layouts/header.php` con logo SENA, título "Sistema Atlas", menú de navegación adaptativo según rol, y botón de logout, crear `layouts/sidebar.php` con navegación lateral que muestre opciones relevantes: admin ve todo, portero solo "Escanear" y "Buscar", usuarios ven "Mis Equipos", implementar `layouts/footer.php` con información del centro y versión del sistema, diseñar estructura de página con CSS Grid: header fijo superior, sidebar lateral colapsable en móvil, área de contenido principal con max-width, footer, implementar diseño responsivo con breakpoints: 768px (tablets), 1024px (desktop), agregar menú hamburguesa para móviles, y crear componentes reutilizables: cards, botones, formularios, tablas con estilos consistentes.

Tarea 6.3 - Aplicar diseño a todas las vistas (Dev 1, 2 días, Media prioridad): Revisar cada vista del sistema aplicando la plantilla base desarrollada, estandarizar formularios: labels alineados, inputs con bordes sutiles, botones con colores apropiados (submit verde, cancel gris, delete rojo), diseñar tablas con estilos: header con fondo verde claro, filas alternadas, hover effects, responsive con scroll horizontal en móvil, crear cards para dashboard: bordes redondeados, sombra sutil, iconos FontAwesome o similares, implementar modals consistentes para confirmaciones y formularios secundarios, agregar breadcrumbs en páginas internas para navegación, y aplicar transiciones CSS suaves en interacciones (hover, focus, transitions).

Tarea 6.4 - Optimizar interfaz móvil (Dev 2, 1 día, Media prioridad): Implementar media queries específicas para dispositivos móviles, ajustar tamaños de fuente para legibilidad en pantallas pequeñas (mínimo 14px), aumentar áreas táctiles de botones y enlaces (mínimo 44×44px como

recomienda Apple), convertir tablas en cards apiladas verticalmente en móvil, hacer formularios de una columna en móviles, agregar menú hamburguesa con animación suave para sidebar, optimizar scanner QR para orientación portrait y landscape, reducir padding y márgenes en móvil manteniendo legibilidad, y probar en dispositivos reales: Android, iOS, tablets.

Tarea 6.5 - Agregar validaciones y feedback (Dev 1, 1 día, Media prioridad): Implementar mensajes de error inline en formularios con iconos y colores rojos, mostrar mensajes de éxito con banners verdes temporales (auto-dismiss tras 3s), agregar loaders/spinners durante operaciones asíncronas (escaneo QR, búsquedas AJAX), implementar confirmaciones con modals elegantes usando SweetAlert2 o similar para acciones críticas (eliminar equipo, cerrar sesión), validar formularios con HTML5: required, pattern, type="email", type="number", agregar validación JavaScript personalizada antes de envío, deshabilitar botón submit tras primer click para prevenir duplicados, mostrar indicadores de campos obligatorios con asteriscos rojos, y implementar tooltips informativos en campos complejos.

Fase 7: Seguridad y Optimización (4.0 días)

Esta fase fortalece la seguridad y el rendimiento:[37] [25] [42]

Tarea 7.1 - Implementar protección CSRF (Dev 2, 1 día, Alta prioridad): Crear función `generateCSRFToken()` que genere token único usando `bin2hex(random_bytes(32))`, almacenar token en `$_SESSION['csrf_token']` al inicio de sesión, desarrollar helper `csrfField()` que retorne `<input type="hidden" name="csrf_token" value="...">`, incluir campo CSRF en todos los formularios del sistema, crear función `validateCSRFToken()` que compare token recibido con el almacenado, rechazar peticiones POST/PUT/DELETE sin token válido con error 403, regenerar token tras uso exitoso o cada X minutos, y documentar implementación en README.

Tarea 7.2 - Sanitizar y validar entradas (Dev 1, 1 día, Alta prioridad): Revisar todos los puntos de entrada de datos del usuario, aplicar `htmlspecialchars()` a toda salida que muestre datos de usuario para prevenir XSS, usar prepared statements con PDO en el 100% de las consultas SQL para prevenir inyección, implementar whitelist de valores permitidos en campos enum (roles, tipos de anomalía), validar tipos de datos: `filter_var()` para emails, expresiones regulares para formatos específicos, sanitizar nombres de archivo antes de guardar con `preg_replace('/[^a-zA-Z0-9_\.]/', '', $filename)`, limitar longitud de campos de texto para prevenir ataques de buffer, validar rangos numéricos (fechas, IDs), y registrar intentos de inyección detectados en log de seguridad.

Tarea 7.3 - Implementar rate limiting (Dev 2, 0.5 días, Media prioridad): Crear tabla `login_attempts` con campos: ip_address, username, timestamp, success, implementar contador de intentos fallidos por IP en ventana de tiempo (ej. 5 intentos en 15 minutos), bloquear temporalmente IP tras exceder límite (30 minutos de bloqueo), almacenar intentos en base de datos o caché (Redis si disponible), mostrar mensaje "Demasiados intentos, intenta en X minutos" con countdown, permitir a admins desbloquear IPs desde panel de administración, considerar implementar CAPTCHA tras 3 intentos fallidos, y registrar IPs bloqueadas en log de seguridad.

Tarea 7.4 - Optimizar consultas SQL (Dev 1, 0.5 días, Media prioridad): Analizar queries más frecuentes con `EXPLAIN` para identificar cuellos de botella, agregar índices compuestos en

columnas frecuentemente filtradas juntas: `INDEX idx_acceso_fecha (id_equipo, fecha_hora)`, crear índices en claves foráneas que aún no los tengan, optimizar JOINs: asegurar que se usan claves indexadas, limitar resultados con `LIMIT` en listados paginados, evitar `SELECT *`, especificar solo columnas necesarias, usar `COUNT(*)` eficientemente con índices, y considerar vistas materializadas para reportes complejos de dashboard.

Tarea 7.5 - Configurar manejo de errores (Dev 2, 1 día, Media prioridad): Configurar `php.ini` para producción: `display_errors = Off`, `log_errors = On`, establecer ruta de log: `error_log = /var/log/php_errors.log`, crear clase `ErrorHandler` con método `handle($errno, $errstr, $errfile, $errline)`, registrar errores con contexto: timestamp, tipo, mensaje, archivo, línea, usuario afectado, implementar diferentes niveles de log: ERROR, WARNING, INFO, DEBUG, rotar logs automáticamente cuando excedan tamaño (ej. 10MB), enviar notificación a admins en errores críticos, crear página de error amigable (error.php) sin exponer detalles técnicos, y capturar excepciones no manejadas con `set_exception_handler()`.

Fase 8: Pruebas y Despliegue (8.0 días)

Esta fase final asegura calidad y prepara producción:[44] [45] [46]

Tarea 8.1 - Realizar pruebas funcionales (Dev 1 y Dev 2, 2 días, Alta prioridad): Crear matriz de pruebas documentando casos: acción, entrada esperada, resultado esperado, probar flujo completo de registro de usuario: validaciones, creación exitosa, email duplicado, probar autenticación: login correcto, contraseña incorrecta, usuario inexistente, sesión persistente, probar registro de equipos: formulario completo, validación de serial único, subida de imágenes, generación de QR, probar módulo de portería: escaneo QR exitoso, búsqueda manual, registro entrada/salida, validaciones de estado, probar detección de anomalías: generar casos conocidos y verificar detección, probar dashboard admin: métricas correctas, filtros funcionando, exportación CSV, probar cada rol por separado verificando permisos apropiados, y documentar bugs encontrados con pasos para reproducir.

Tarea 8.2 - Pruebas de integración (Dev 1 y Dev 2, 1 día, Alta prioridad): Probar flujo end-to-end completo: Usuario se registra → Registra equipo → Genera QR → Portero escanea → Registra entrada → Usuario verifica estado → Portero registra salida → Admin revisa métricas, verificar integridad referencial: eliminar usuario con equipos (debe fallar o cascadear), eliminar equipo con registros (verificar comportamiento), probar sincronización entre roles: cambios en admin reflejados inmediatamente para usuarios, simular múltiples usuarios concurrentes realizando acciones simultáneas, verificar transacciones: rollback en caso de errores parciales, probar recuperación ante fallos: base de datos no disponible, servidor caído, y validar consistencia de datos tras operaciones CRUD completas.

Tarea 8.3 - Corregir bugs (Dev 1 y Dev 2, 2 días, Alta prioridad): Priorizar bugs por severidad: críticos (impiden uso), altos (afectan funcionalidad principal), medios, bajos, reproducir cada bug documentado paso a paso, implementar fixes aplicando mejores prácticas, probar fix en ambiente de desarrollo, verificar que el fix no introduce regresiones en funcionalidad existente, actualizar documentación si el bug revela comportamiento inesperado, re-probar flujos completos tras cada fix importante, cerrar tickets de bugs verificando resolución, y mantener changelog de cambios realizados.

Tarea 8.4 - Crear documentación técnica (Dev 1, 1 día, Media prioridad): Escribir README.md completo con: descripción del proyecto, requisitos del sistema (PHP 8.2+, MySQL 8.0, Docker), instrucciones de instalación paso a paso con comandos exactos, configuración de variables de entorno (.env.example), estructura del proyecto explicada, guía de desarrollo: cómo agregar modelos, controladores, vistas, documentar arquitectura MVC y patrones utilizados, listar endpoints principales de la API interna, incluir diagrama de base de datos (ERD), explicar sistema de roles y permisos, documentar librerías externas utilizadas y sus versiones, agregar sección de troubleshooting con problemas comunes, y mantener actualizado con cada cambio significativo.

Tarea 8.5 - Crear manual de usuario (Dev 2, 1 día, Media prioridad): Redactar guías separadas por rol: porteros (cómo escanear, buscar, registrar), usuarios (cómo registrarse, agregar equipos, ver estado), administradores (gestión completa), incluir capturas de pantalla de cada interfaz principal, escribir tutoriales paso a paso para tareas comunes: "Cómo registrar mi laptop", "Cómo escanear un QR", "Cómo resolver una anomalía", agregar sección de preguntas frecuentes (FAQ), explicar mensajes de error comunes y soluciones, documentar flujos de casos especiales: equipo reportado, acceso fuera de horario, crear versión PDF descargable del manual, y versión HTML integrada en el sistema con búsqueda.

Tarea 8.6 - Preparar entorno de producción (Dev 1, 1 día, Alta prioridad): Configurar servidor de producción: instalar Docker y Docker Compose, clonar repositorio desde GitHub en servidor, configurar variables de entorno de producción (.env con credenciales reales), establecer permisos de archivos correctos (755 directorios, 644 archivos, 777 uploads y logs), configurar SSL/TLS con Let's Encrypt para HTTPS obligatorio, ajustar php.ini de producción: desactivar display_errors, aumentar max_upload_size para imágenes, configurar backup automático de base de datos: cron job diario con mysqldump, implementar rotación de backups (mantener últimos 30 días), configurar firewall: solo puertos 80, 443 y SSH abiertos, configurar dominio DNS apuntando al servidor, y realizar prueba de carga básica con herramientas como Apache Bench.

Recomendaciones Finales

Gestión del Proyecto

Para maximizar el éxito del proyecto Atlas, se recomienda implementar las siguientes prácticas de gestión:[44] [45] [47] [49]

Herramientas de colaboración: Utilizar Trello o Asana para seguimiento de tareas del plan de trabajo, dividiendo el tablero en columnas (Por Hacer, En Progreso, Bloqueado, Completado). Asignar cada tarea específica del plan a Dev 1 o Dev 2 con fechas límite claras.[19] [20] [21]

Control de versiones: Establecer workflow de Git con ramas: `main` para código estable de producción, `develop` para integración de nuevas funcionalidades, `feature/nombre-tarea` para cada tarea específica del plan. Dev 1 y Dev 2 deben hacer commits frecuentes con mensajes descriptivos siguiendo convención (ej. `feat: implementar login`, `fix: corregir validación de serial`) y crear pull requests para revisión mutua antes de fusionar a develop.[43] [44]

Reuniones de sincronización: Realizar daily standups breves (15 minutos) al inicio de cada jornada donde cada desarrollador comparte qué hizo ayer, qué hará hoy, y si tiene bloqueos. Al finalizar cada fase del plan, realizar retrospectiva para identificar qué funcionó bien, qué mejorar, y ajustar estimaciones de fases restantes.[47] [44]

Documentación continua: No dejar la documentación para el final. A medida que se completa cada tarea, actualizar README con funcionalidades implementadas, decisiones técnicas tomadas, y configuraciones necesarias. Esto facilitará la fase 8 de pruebas y documentación.[45] [47]

Priorización y Gestión de Riesgos

El plan de trabajo está diseñado con prioridades específicas (Alta, Media, Baja) que reflejan dependencias y criticidad. Se recomienda:[44] [47]

Respeto de prioridades: Completar todas las tareas de Alta prioridad de una fase antes de avanzar a tareas de Media prioridad. Las tareas marcadas como Alta prioridad son fundamentales para funcionalidades críticas o son prerequisitos de otras tareas.

Buffer para imprevistos: Las estimaciones incluyen tiempo de aprendizaje apropiado para aprendices SENA, pero pueden surgir imprevistos. Mantener un buffer del 20% en el calendario total (agregar 1-2 semanas adicionales) para investigación extra, corrección de bugs inesperados, o retrasos por factores externos.[45] [46]

Identificación temprana de bloqueos: Si una tarea de Alta prioridad encuentra problemas técnicos significativos, elevar inmediatamente al instructor del SENA o buscar ayuda en comunidades especializadas (Stack Overflow, foros PHP) en lugar de bloquearse días completos. El tiempo es crítico en proyectos con plazos fijos.

Seguridad como Prioridad Transversal

Aunque la Fase 7 se dedica específicamente a seguridad, este debe ser un tema presente desde el inicio:[37] [25] [42]

Durante desarrollo: Al escribir cada controlador y modelo, aplicar inmediatamente prepared statements PDO, escapar toda salida con `htmlspecialchars()`, y validar tipos de datos recibidos. Es más fácil implementar seguridad desde el inicio que refactorizar código inseguro después.

Revisión de código de seguridad: En las pull requests, el revisor debe verificar específicamente que no haya consultas SQL concatenadas directamente, que las contraseñas siempre se hasheen con `password_hash()`, que los formularios incluyan tokens CSRF, y que no se exponga información sensible en mensajes de error.

Pruebas de penetración básicas: Al finalizar cada fase, intentar explotar vulnerabilidades comunes: inyección SQL en formularios, XSS en campos de texto, acceso a rutas sin autenticación, manipulación de IDs en URLs, CSRF en acciones críticas. Herramientas como OWASP ZAP pueden automatizar parte de estas pruebas.[42] [37]

Aprendizaje y Recursos

Como proyecto de formación SENA, es fundamental aprovechar los recursos disponibles:[5] [6] [7]

Documentación oficial: Consultar documentación oficial de PHP (php.net), MySQL (dev.mysql.com), y las librerías utilizadas antes de buscar tutoriales externos. La documentación oficial suele estar más actualizada y ser más precisa.

Comunidad de desarrollo: Unirse a comunidades hispanohablantes como r/programacion en Reddit, grupos de PHP en Discord, o foros como Stack Overflow en Español. Hacer preguntas específicas mostrando código y describiendo lo intentado.

Instructores y pares SENA: Aprovechar sesiones de asesoría con instructores del programa de Análisis y Desarrollo de Software del SENA. Compartir aprendizajes con compañeros de formación que puedan estar trabajando en proyectos similares.

Registro de aprendizajes: Mantener un documento de "lecciones aprendidas" donde ambos desarrolladores anoten soluciones a problemas complejos, conceptos nuevos dominados, y recursos útiles encontrados. Esto será valioso para proyectos futuros y como portafolio de aprendizaje.

Escalabilidad Futura

Aunque el MVP se enfoca en funcionalidad simple y rápida implementación, diseñar pensando en evolución futura:[4] [36]

Arquitectura modular: La estructura MVC propuesta permite agregar funcionalidades sin reestructurar todo el sistema. Por ejemplo, agregar un módulo de reportes avanzados o integración con sistemas externos del SENA.

Abstracción de lógica de negocio: Separar validaciones complejas en clases dedicadas (ej. `EquipoValidator`, `AccesoValidator`) facilita modificar reglas de negocio sin tocar controladores.

Configuración flexible: Usar tabla `configuracion_horario` en lugar de valores hardcodeados permite que administradores ajusten parámetros del sistema sin intervención técnica. Considerar crear tabla general `configuraciones` para otros parámetros (timeout sesión, tamaño máximo imágenes, etc.).

API para integraciones: Aunque no es prioritario en el MVP, estructurar controladores pensando en que podrían retornar JSON además de renderizar vistas facilita crear API REST futura para integración con app móvil o sistemas externos.

Conclusión

El Sistema Atlas representa una oportunidad significativa para modernizar el control de acceso de equipos en el Centro Tecnológico de la Amazonia del SENA, sustituyendo procesos manuales ineficientes por una plataforma digital funcional y escalable. El proyecto está cuidadosamente dimensionado para ser completado en aproximadamente 7-8 semanas por dos aprendices del

programa de Análisis y Desarrollo de Software, priorizando tecnologías nativas que minimizan la curva de aprendizaje mientras entregan valor real.[1] [4] [36] [5] [6]

La arquitectura MVC simplificada, combinada con las mejores prácticas de seguridad y el diseño de base de datos normalizado 3NF, proporciona una base sólida que no solo cumplirá con los requisitos inmediatos del centro, sino que también servirá como experiencia formativa valiosa para los desarrolladores. El plan de trabajo detallado con distribución equilibrada de tareas entre Dev 1 y Dev 2 asegura que ambos aprendices desarrollen competencias tanto en backend como en frontend, preparándolos para desafíos profesionales futuros.[2] [30] [44] [31] [45] [47]

Los principales factores de éxito del proyecto serán: adherencia estricta al plan de trabajo con priorización de tareas críticas, implementación de seguridad desde el inicio en lugar de como añadido posterior, comunicación constante entre los desarrolladores mediante herramientas de colaboración modernas, y aprovechamiento de los recursos de formación del SENA incluyendo asesorías con instructores. Al completar este proyecto, no solo se habrá entregado una herramienta útil para la institución, sino que los dos aprendices habrán consolidado conocimientos prácticos en desarrollo full-stack, gestión de proyectos, trabajo en equipo y resolución de problemas reales, competencias fundamentales para su carrera profesional en tecnología.[37] [25] [42]

⁂

1. https://stackoverflow.com/questions/19648760/project-structure-for-php
2. https://dev.to/pranav_aadithya_36edf63cb/database-normalization-in-mysql-1nf-2nf-3nf-simple-example-314p
3. https://dev.to/amirezaeb/the-ultimate-php-qr-code-library-3blf
4. https://brainstreamtechnolabs.com/professional-php-development-coding-standards-best-practices-performance/
5. https://www.noticiascaracol.com/estilo-de-vida/sena-abrio-inscripciones-al-curso-virtual-en-desarrollo-de-software-uno-de-los-mas-solicitados-so35
6. https://zajuna.sena.edu.co/pdfs/titulada/tecgnologias/analisis_desarrollo_software.pdf
7. https://betowa.sena.edu.co/oferta/analisis-y-desarrollo-de-software?programId=136456
8. https://github.com/hamza1886/role-based-access-control
9. https://www.sitepoint.com/role-based-access-control-in-php/
10. https://www.geeksforgeeks.org/php/dynamically-generating-a-qr-code-using-php/
11. https://phpqrcode.sourceforge.net
12. https://www.sitepoint.com/generate-qr-codes-in-php/
13. https://saasdev.hashnode.dev/how-to-create-a-qr-code-scanner-in-php
14. https://www.dynamsoft.com/codepool/php-laravel-barcode-qr-code-reader.html
15. https://www.geeksforgeeks.org/javascript/create-a-qr-code-scanner-or-reader-in-html-css-javascript/
16. https://github.com/msaf9/student-attendance-management-system
17. https://stackoverflow.com/questions/3939764/what-is-a-good-database-design-schema-for-a-attendance-database

18. https://five.co/blog/create-an-attendance-database/
19. https://tcservi.com/mejores-herramientas-gestion-proyectos/
20. https://slack.com/intl/es-es/blog/collaboration/herramientas-de-gestion-de-proyectos
21. https://asana.com/es/resources/best-project-management-software
22. https://www.red-gate.com/blog/normalize-2nf-3nf
23. https://www.digitalocean.com/community/tutorials/database-normalization
24. https://popsql.com/blog/normalization-in-sql
25. https://mrebi.com/es/php/php-security-practices/
26. https://www.sitepoint.com/docker-php-development-environment/
27. https://dev.to/truthseekers/setup-a-basic-local-php-development-environment-in-docker-kod
28. https://github.com/chillerlan/php-qrcode
29. https://www.youtube.com/watch?v=xgFu26FWx5Y
30. https://200oksolutions.com/blog/building-custom-mvc-framework-core-php-architecture-best-practices/
31. https://www.sitepoint.com/the-mvc-pattern-and-php-1/
32. https://github.com/andrejrs/php-mvc-example
33. https://phptherightway.com
34. https://www.freecodecamp.org/news/create-an-mvc-framework-from-scratch-with-php/
35. https://stackoverflow.com/questions/55662469/php-rest-how-do-i-implement-a-role-based-access-of-my-api
36. https://accesto.com/blog/evaluating-modern-php/
37. https://tuxcare.com/es/blog/php-vulnerability/
38. https://reliasoftware.com/blog/php-project-ideas
39. https://dev.to/hunterdev/what-can-you-build-with-php-in-2025-more-than-you-think-1edi
40. https://blog.minhazav.dev/research/html5-qrcode
41. https://aws.plainenglish.io/how-to-set-up-a-php-development-environment-using-docker-74b86160749c
42. https://dcreations.es/blog/php/mejores-practicas-seguridad-php
43. https://github.com/jlucki/docker-php-dev-env
44. https://www.usemotion.com/blog/software-project-planning.html
45. https://ddi-dev.com/blog/it-news/software-development-project-plan-a-full-guide-2024/
46. https://www.invensislearning.com/blog/a-guide-to-software-project-management-phases-best-practices/
47. https://www.itransition.com/software-development/project-plan
48. https://developer.auth0.com/resources/code-samples/api/laravel/basic-role-based-access-control
49. https://www.projectmanager.com/guides/project-planning
50. https://www.geeksforgeeks.org/dbms/third-normal-form-3nf/
51. https://www.youtube.com/watch?v=8xPWPGxL7Xk
52. https://chat2db.ai/resources/blog/3nf-in-dbms

53. https://www.itservicesindia.com/blog/php-development-trends-strategies-2025
54. https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/
55. https://www.teachngo.com/blog/how_to_create_a_student_attendance_database
56. https://www.digitalocean.com/community/meetup-kits/how-to-create-php-development-environments-with-docker-compose-a-digitalocean-workshop-kit
57. https://www.youtube.com/watch?v=VLBp3pJFSQc
58. https://www.softr.io/create/attendance-tracking-system
59. https://docs.docker.com/guides/php/develop/
60. https://es.scribd.com/document/563408255/rbac-role-based-access-control-slides
61. https://dev.to/yogaraj/building-my-first-sql-project-employee-attendance-leave-management-system-30m4
62. https://phprbac.net/docs_tutorial.php
63. https://github.com/libern/qr-code-reader
64. https://www.youtube.com/watch?v=86wsJgyf7G4
65. https://stackoverflow.com/questions/3899631/how-can-you-read-qr-codes-in-php
66. https://www.giuseppemaccario.com/best-practices-for-mvc-architecture-in-php-common-mistakes-and-how-to-avoid-them-part-1-controllers/
67. https://stafiz.com/en/how-to-make-a-step-by-step-project-schedule/
68. https://theonetechnologies.com/portfolio/qr-code-scanner-app
69. https://dev.to/ulisesafcdev/desarrollar-un-proyecto-mvc-con-php-puro-4akg
70. https://translate.google.com/translate?u=https%3A%2F%2Fwpwebinfotech.com%2Fblog%2Fphp-security%2F&hl=es&sl=en&tl=es&client=srp
71. https://www.wradio.com.co/2025/02/25/cuanto-dura-la-carrera-de-desarrollo-de-software-en-sena-equivalencia-anos-y-nombre-del-tecnologo/
72. https://www.wrike.com/es/project-management-guide/faq/que-son-las-herramientas-de-gestion-de-proyectos/
73. https://kinsta.com/es/blog/10-cosas-que-no-se-deben-hacer-en-php7/
74. https://metalmecanicosena.blogspot.com/p/analisis-y-desarrollo-de-software.html
75. https://businessmap.io/es/herramientas-de-software-gestion-de-proyectos
76. https://www.q2bstudio.com/nuestro-blog/49100/el-mejor-marco-de-php-para-desarrollo-web-en-2025
77. https://betowa.sena.edu.co/oferta/calidad-en-el-desarrollo-de-software?programId=70957
78. https://www.flowlu.com/es/blog/project-management/best-freeware-project-management-software/
79. https://keepcoding.io/blog/futuro-de-php-y-a-que-se-debe-su-popularidad/
80. https://www.youtube.com/watch?v=bEsi8FxkPEA