Documentación Técnica

Plataforma Web para el Control y Gestión del Servicio Médico UNEARTE

Autores:

Alcides Mata C.I: V- 31.158.186

Giannefran Radomile C.I: V- 28.653.684

Joyce Vadacchino C.I: V- 17.146.941

Mijael Engelmann C.I: V- 31.222.463

Yannis Iturriago C.I: V- 29.797.308

15/10/2025

ÍNDICE

1. Introducción

El presente documento describe la arquitectura técnica y el diseño de la Plataforma Web para el Control y Gestión del Servicio Médico de la Universidad Nacional Experimental de las Artes (UNEARTE). Este sistema fue desarrollado como parte de un proyecto de servicio comunitario adscrito a la Universidad Nacional Experimental de las Telecomunicaciones e Informática (UNETI), en cumplimiento con la Ley de Servicio Comunitario del Estudiante de Educación Superior de Venezuela.

El propósito fundamental del sistema es modernizar y optimizar los procesos administrativos y de atención del servicio médico de UNEARTE, que previamente operaban de forma manual y descentralizada. La dependencia de registros en papel generaba ineficiencias, riesgo de pérdida de datos y dificultades en el seguimiento de historiales clínicos.

En respuesta a esta problemática, el objetivo general del proyecto fue "Desarrollar una plataforma web funcional que permita el control y gestión integral del servicio médico de la UNEARTE". Los objetivos específicos incluyeron diagnosticar necesidades, diseñar la solución, implementar módulos clave (pacientes, citas, historiales, inventario, reportes) y capacitar al personal.

Desarrollado sobre el framework Django y utilizando PostgreSQL como gestor de base de datos, el sistema centraliza el registro de pacientes, la gestión de citas médicas, el manejo de historiales clínicos, la administración del inventario farmacéutico y la generación de reportes. Incorpora además un sistema de autenticación basado en roles para controlar el acceso a las diferentes funcionalidades.

Esta documentación técnica está dirigida al personal técnico de UNEARTE responsable del mantenimiento y futuras evoluciones del sistema, así como a la UNETI para fines de evaluación académica. Detalla la arquitectura, el modelo de datos, la lógica de negocio, el sistema de autenticación y el entorno operativo de la plataforma.

2. Arquitectura del Sistema

La plataforma se basa en el framework **Django**, un marco de desarrollo web de alto nivel escrito en Python que sigue el patrón arquitectónico Modelo-Vista-Template (MVT). Esta elección facilita un desarrollo rápido, seguro y mantenible. La arquitectura modular de Django permite organizar el proyecto en aplicaciones autocontenidas, cada una responsable de una funcionalidad específica del sistema.

El archivo sistema_medico/settings.py define la configuración central del proyecto, incluyendo las aplicaciones instaladas (INSTALLED_APPS). El archivo sistema_medico/urls.py actúa como el enrutador principal, delegando las rutas a las aplicaciones correspondientes.

Las aplicaciones que componen el sistema son:

2.1. Aplicación core

- **Propósito:** Gestiona las funcionalidades transversales, la autenticación, la gestión de usuarios y roles, y la estructura general del sitio. Incluye el *Dashboard* principal, páginas estáticas ('Acerca de', 'Contacto'), manejo de errores (404, 500), búsqueda global y la plantilla base (base.html). Implementa un sistema de roles (admin, medico, recepcionista) a través del modelo PerfilUsuario y utiliza decoradores/mixins para el control de acceso.
- **Archivos Clave:**
    - models.py: Define PerfilUsuario con roles y relación OneToOneField con el usuario de Django (User).
    - views.py: Incluye vistas para el *Dashboard* (DashboardView), páginas estáticas (AboutView, ContactView), registro (signup), perfil de usuario (profile), gestión de usuarios (CRUD), cambio de contraseña y rol, búsqueda global (search_all), y manejo de errores (handler404, handler500).
    - forms.py: Contiene formularios para registro (SignUpForm), actualización de usuario (UserUpdateForm), actualización de perfil (ProfileUpdateForm), cambio de rol (UserRoleForm), y contraseña (CustomPasswordChangeForm, CustomSetPasswordForm).
    - urls.py: Define rutas para el *Dashboard*, páginas estáticas, perfil, gestión de usuarios, búsqueda global y acceso denegado.
    - decorators.py: Implementa decoradores (@admin_required, @personal_medico_required, @recepcionista_required) para restringir el acceso a vistas basado en roles.
    - signals.py: Utiliza señales de Django (post_save) para crear o actualizar

automáticamente un PerfilUsuario cuando se crea o modifica un User.

- templates/core/: Plantillas específicas de la aplicación.
- templates/registration/: Plantillas para login, logout, registro y gestión de usuarios.

- templates/base.html: Plantilla maestra con la estructura principal (navegación, footer, etc.).

2.2. Aplicación pacientes

- **Propósito:** Encargada de la gestión de la información demográfica y de contacto de los pacientes. Permite operaciones CRUD para pacientes, incluyendo datos básicos, dirección y teléfonos. Se ha simplificado para asumir Cédula de Identidad como único tipo de documento y una estructura de dirección más específica para UNEARTE (Caracas). Incluye funcionalidad AJAX para la creación modal de TipoTelefono.
- **Archivos Clave:**
  - models.py: Define Paciente, Genero (usando TextChoices), Pais, Estado, Ciudad, TipoTelefono, Direccion (simplificada con OneToOneField), y Telefono.
  - views.py: Vistas CRUD (index, create, show, edit, destroy), búsqueda (search), exportación a PDF/Excel (export_pacientes_pdf, export_pacientes_excel), y vistas AJAX para crear TipoTelefono (crear_tipo_telefono_ajax). Utiliza decoradores de roles para control de acceso.
  - forms.py: PacienteForm, DireccionForm, TelefonoForm, TipoTelefonoForm. Incluye validaciones específicas para cédula y teléfono venezolano.
  - urls.py: Rutas CRUD, búsqueda, exportaciones y AJAX.
  - templates/pacientes/: Plantillas para listar, crear, ver, editar, buscar pacientes y plantillas PDF/modales.

2.3. Aplicación citas

- **Propósito:** Gestiona la programación, seguimiento y registro de citas médicas. Permite agendar citas asociadas a pacientes, definir tipos, motivos y estados. Incluye la gestión de TipoCita, MotivoCita, EstadoCita como entidades separadas con sus propias vistas CRUD. Implementa funcionalidad AJAX para crear estas entidades relacionadas modalmente y para cambiar el estado de la cita. Permite la exportación de citas a PDF/Excel.

- **Archivos Clave:**
    - models.py: Define EstadoCita, TipoCita, MotivoCita, Cita, TipoNota, NotaCita.
    - views.py: Vistas CRUD para Cita, EstadoCita, TipoCita, MotivoCita. Vistas para citas_hoy, search, cambiar_estado, exportaciones (export_citas_pdf, export_citas_excel), y vistas AJAX (crear_tipo_cita_ajax, crear_motivo_cita_ajax, crear_estado_cita_ajax, cambiar_estado_cita_ajax). Utiliza decoradores de roles.
    - forms.py: CitaForm, EstadoCitaForm, TipoCitaForm, MotivoCitaForm.
    - urls.py: Rutas CRUD para citas y entidades relacionadas, búsqueda, citas_hoy, cambio de estado, exportaciones y AJAX.
    - templates/citas/: Plantillas para citas, estados, motivos, tipos, y plantillas PDF/modales.

2.4. Aplicación historiales

- **Propósito:** Administra la información clínica de los pacientes (alergias, enfermedades preexistentes, medicamentos actuales). Se vincula OneToOne con Paciente. Los campos clínicos ahora usan relaciones ManyToManyField con los nuevos modelos Alergia y Enfermedad para una gestión estructurada. Permite la creación modal de estas entidades relacionadas y la exportación de historiales a PDF/Excel.
- **Archivos Clave:**
    - models.py: Define Alergia, Enfermedad, HistorialMedico (con campos ManyToManyField).
    - views.py: Vistas CRUD para HistorialMedico. Vistas de búsqueda (search), exportaciones (export_historiales_pdf, export_historiales_excel), y vistas AJAX para crear Alergia y Enfermedad (crear_alergia_ajax, crear_enfermedad_ajax). Utiliza decoradores de roles.
    - forms.py: HistorialMedicoForm, AlergiaForm, EnfermedadForm.
    - urls.py: Rutas CRUD, búsqueda, exportaciones y AJAX.
    - templates/historiales/: Plantillas para historiales y plantillas PDF/modales.

2.5. Aplicación inventario

- **Propósito:** Gestiona el inventario farmacéutico: categorías, proveedores, medicamentos, existencias (lotes) y movimientos (entradas/salidas). Permite la creación modal de Categoria, Proveedor y Medicamento. Incluye vistas para

consultar el stock actual y el historial de movimientos. Permite la exportación de datos del inventario a PDF/Excel.

- **Archivos Clave:**
  - models.py: Define Categoria, Proveedor, Medicamento, Inventario, MovimientoInventario. El cálculo de stock_actual en Medicamento se basa ahora en MovimientoInventario.
  - views.py: Vistas CRUD para Categoria, Proveedor, Medicamento, Inventario. Vistas para listar movimientos (listar_movimientos), registrar salidas (crear_salida), ver stock (stock_medicamentos), dashboard del inventario (index), exportaciones (export_categorias_pdf, export_proveedores_excel, etc.), y vistas AJAX (crear_categoria_ajax, crear_proveedor_ajax, crear_medicamento_ajax). Utiliza decoradores de roles.
  - forms.py: CategoriaForm, ProveedorForm, MedicamentoForm, InventarioForm, MovimientoInventarioForm (para salidas).
  - urls.py: Rutas CRUD para cada entidad, movimientos, stock, exportaciones y AJAX.
  - templates/inventario/: Plantillas para cada entidad, movimientos, stock, y plantillas PDF/modales.

3. Modelo de Datos (Base de Datos)

La persistencia de los datos se gestiona a través del ORM (Object-Relational Mapper) de Django, que mapea los modelos definidos en Python a tablas en una base de datos relacional. La base de datos configurada para este proyecto es **PostgreSQL** (sistema_medico/settings.py), elegida por su robustez, escalabilidad y características avanzadas, adecuada para aplicaciones de gestión de datos sensibles como la información médica.

3.1. Diagrama Entidad-Relación (Descripción Conceptual)

El esquema se organiza en torno a:

- **Usuarios y Roles (core):** El modelo User de Django se extiende con PerfilUsuario (1-1) para asignar roles (admin, medico, recepcionista).
- **Pacientes (pacientes):** Paciente es central, con relaciones N-1 a Genero y 1-1 a Direccion. Direccion tiene relaciones N-1 a Ciudad, Estado, Pais. Telefono tiene relación N-1 a Paciente y TipoTelefono. (Se asume Cédula como único tipo de documento).
- **Historiales (historiales):** HistorialMedico tiene relación 1-1 con Paciente y relaciones N-N con Alergia, Enfermedad y Medicamento (del app inventario).
- **Citas (citas):** Cita tiene relaciones N-1 con Paciente, TipoCita, MotivoCita, EstadoCita. NotaCita tiene relación N-1 con Cita y TipoNota.
- **Inventario (inventario):** Medicamento tiene relaciones N-1 con Categoria y Proveedor. Inventario (existencia) tiene relación N-1 con Medicamento. MovimientoInventario tiene relación N-1 con Medicamento y User (implícito por usuario=request.user en la vista).

Esta estructura sigue principios de normalización para minimizar la redundancia y asegurar la integridad de los datos.

3.2. Descripción Detallada de Modelos

A continuación, se detallan las tablas generadas por los modelos de Django:

Aplicación core (models.py)

- **Modelo PerfilUsuario (core_perfilusuario)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
user	OneToOneField(User)	Relación uno a uno con el modelo de usuario de Django.
rol	CharField(max_length=15)	Rol del usuario ('admin', 'medico', 'recepcionista'). Default: 'recepcionista'.
avatar	ImageField	Campo para la imagen de perfil del usuario (opcional).

Aplicación pacientes (models.py)

- **Modelo Genero (generos)** (Usando TextChoices)

Campo	**Tipo**	**Descripción**
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(max_length=10)	Nombre del género (MASCULINO, FEMENINO).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo Pais (paises)**

Campo	**Tipo**	**Descripción**
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(100)	Nombre único del país.
codigo_iso	CharField(3)	Código ISO 3166-1 alfa-3 único del país.
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo Estado (estados)** (Renombrado desde Departamento)

Campo	**Tipo**	**Descripción**
id	BigAutoField	Clave primaria autoincremental.

nombre	CharField(100)	Nombre del estado/entidad federal.
pais	ForeignKey(Pais)	País al que pertenece (Relación N-1).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.
Constraint: unique_together = ('nombre', 'pais')		

- **Modelo Ciudad (ciudades)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(100)	Nombre de la ciudad.
estado	ForeignKey(Estado)	Estado al que pertenece (Relación N-1).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.
Constraint: unique_together = ('nombre', 'estado')		

- **Modelo TipoTelefono (tipos_telefono)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(50)	Nombre único del tipo (Ej: Móvil, Casa).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo Paciente (pacientes)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.

numero_documento	CharField(max_length=8)	Número de Cédula de Identidad (único).
nombre	CharField(max_length=100)	Nombre(s) del paciente.
apellido	CharField(max_length=100)	Apellido(s) del paciente.
fecha_nacimiento	DateField	Fecha de nacimiento del paciente.
genero	CharField(max_length=10)	Género del paciente (usando Genero.choices).
email	EmailField(max_length=100)	Correo electrónico (opcional, único).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.
Constraints: unique en numero_documento, unique en email. Índices en (apellido, nombre) y fecha_nacimiento. @property edad.		

- **Modelo Direccion (direcciones)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
paciente	OneToOneField(Paciente)	Paciente al que pertenece la dirección (1-1).
pais	ForeignKey(Pais)	País (default=Venezuela).

estado	ForeignKey(Estado)	Estado (default=Distrito Capital).
ciudad	ForeignKey(Ciudad)	Ciudad (default=Caracas).
direccion	CharField(255)	Dirección detallada (calle, parroquia, etc.).
codigo_postal	CharField(4)	Código postal (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo Telefono (telefonos)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
paciente	ForeignKey(Paciente)	Paciente al que pertenece el teléfono (N-1).
tipo_telefono	ForeignKey(TipoTelefono)	Tipo de teléfono (Ej: Móvil, Casa) (N-1).
numero	CharField(11)	Número de teléfono (formato Vzla 11 dígitos).
es_principal	BooleanField	Indica si es el teléfono principal (default=False).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.
Constraint: unique_together = ('paciente', 'numero')		

Aplicación historiales (models.py)

- **Modelo Alergia (historiales_alergia)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(100)	Nombre único de la alergia.
descripcion	TextField	Descripción adicional (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo Enfermedad (historiales_enfermedad)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(150)	Nombre único de la enfermedad preexistente.
descripcion	TextField	Descripción adicional (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo HistorialMedico (historiales_medicos)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria

		autoincremental.
paciente	OneToOneField(Paciente)	Paciente al que pertenece el historial (Relación 1-1).
alergias	ManyToManyField(Alergia)	Alergias conocidas del paciente (N-N).
enfermedades_preexistentes	ManyToManyField(Enfermedad)	Enfermedades preexistentes del paciente (N-N).
medicamentos_actuales	ManyToManyField(Medicamento)	Medicamentos que toma actualmente el paciente (N-N).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

Aplicación citas (models.py)

- **Modelo EstadoCita (estados_cita)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(50)	Nombre único del estado (Ej: Pendiente, Confirmada).
descripcion	TextField	Descripción adicional (opcional).
color	CharField(7)	Código de color HEX para UI (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo TipoCita (tipos_cita)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(100)	Nombre único del tipo de cita (Ej: Consulta General).
descripcion	TextField	Descripción adicional (opcional).
duracion_estimada	IntegerField	Duración estimada en minutos (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo MotivoCita (motivos_cita)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(100)	Nombre único del motivo (Ej: Chequeo de rutina).
descripcion	TextField	Descripción adicional (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo Cita (citas)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
paciente	ForeignKey(Paciente)	Paciente asociado a la cita (N-1).
tipo_cita	ForeignKey(TipoCita)	Tipo de la cita (N-1).
motivo	ForeignKey(MotivoCita)	Motivo de la cita (N-1).
fecha	DateField	Fecha de la cita.
hora_inicio	TimeField	Hora de inicio de la cita.
hora_fin	TimeField	Hora de fin de la cita.
estado	ForeignKey(EstadoCita)	Estado actual de la cita (N-1).
observaciones	TextField	Observaciones generales (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.
Constraints: unique_together = ('paciente', 'fecha', 'hora_inicio'), Índices en (fecha, hora_inicio) y estado.		

- **Modelo TipoNota (tipos_nota)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(50)	Nombre único del tipo de nota (Ej: Sistema, Médico).
descripcion	TextField	Descripción adicional (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo NotaCita (notas_cita)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
cita	ForeignKey(Cita)	Cita a la que pertenece la nota (N-1).
tipo_nota	ForeignKey(TipoNota)	Tipo de la nota (N-1).
contenido	TextField	Contenido textual de la nota.
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

Aplicación inventario (models.py)

- **Modelo Categoria (inventario_categorias)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(100)	Nombre único de la categoría (Ej: Analgésicos).
descripcion	TextField	Descripción adicional (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo Proveedor (inventario_proveedores)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(200)	Nombre del proveedor.
contacto	CharField(100)	Nombre de la persona de contacto (opcional).
telefono	CharField(20)	Teléfono de contacto (opcional).
email	EmailField	Correo electrónico (opcional).
direccion	TextField	Dirección física (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

- **Modelo Medicamento (inventario_medicamentos)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
nombre	CharField(200)	Nombre del medicamento.
descripcion	TextField	Descripción adicional (opcional).
categoria	ForeignKey(Categoria)	Categoría a la que pertenece (N-1).
proveedor	ForeignKey(Proveedor)	Proveedor principal (N-1).
codigo	CharField(50)	Código único del medicamento (autogenerado si vacío).
precio_unitario	DecimalField(10, 2)	Precio por unidad.
stock_minimo	IntegerField	Nivel mínimo de stock antes de alerta (default=0).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.
Lógica especial: El método save() genera codigo único. @property stock_actual (calculado con MovimientoInventario), @property estado_stock.		

- **Modelo Inventario (inventario_existencias)** (Representa una existencia/lote específico)

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.

medicamento	ForeignKey(Medicamento)	Medicamento al que pertenece esta existencia (N-1).
cantidad	IntegerField	Cantidad de unidades en esta existencia.
fecha_caducidad	DateField	Fecha de caducidad de este lote.
lote	CharField(50)	Identificador del lote (opcional).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.
Propiedades: @property estado (disponible, caducado, agotado, bajo_stock), @property dias_para_caducar.		

- **Modelo MovimientoInventario (inventario_movimientos)**

Campo	Tipo	Descripción
id	BigAutoField	Clave primaria autoincremental.
medicamento	ForeignKey(Medicamento)	Medicamento afectado por el movimiento (N-1).
tipo	CharField(10)	Tipo de movimiento ('entrada' o 'salida').
cantidad	PositiveIntegerField	Cantidad de unidades movidas (siempre positiva).
fecha	DateTimeField	Fecha y hora del movimiento (auto_now_add=True).
descripcion	TextField	Descripción adicional del movimiento (opcional).
usuario	ForeignKey(User)	Usuario que realizó el movimiento (N-1).
created_at	DateTimeField	Fecha y hora de creación del registro.
updated_at	DateTimeField	Fecha y hora de última actualización.

4. Lógica de Negocio (Flujo de Datos)

La lógica de negocio se implementa principalmente en las vistas (views.py) y formularios (forms.py) de cada aplicación. Las vistas procesan las solicitudes HTTP, interactúan con los modelos para acceder o modificar datos, utilizan los formularios para validar la entrada del usuario y finalmente renderizan plantillas HTML (templates/) para generar la respuesta. Las URLs (urls.py) mapean las rutas de las solicitudes a las vistas correspondientes y el control de acceso basado en roles se aplica mediante decoradores y mixins.

4.1. Autenticación y Gestión de Usuarios (core)

- **Registro (signup):** Permite a nuevos usuarios registrarse. Utiliza SignUpForm. Al guardar, la señal post_save crea un PerfilUsuario asociado.
- **Login/Logout:** Utiliza las vistas incorporadas de Django (LoginView, LogoutView) con plantillas personalizadas.
- **Gestión de Usuarios (CRUD):** Vistas protegidas para administradores (@admin_required) que permiten listar, crear (UserCreationForm), editar (UserUpdateForm, ProfileUpdateForm), eliminar usuarios y cambiar contraseñas (CustomSetPasswordForm).
- **Gestión de Roles (cambiar_rol):** Vista protegida (@admin_required) que permite a un administrador cambiar el rol (PerfilUsuario.rol) de otro usuario usando UserRoleForm.
- **Perfil de Usuario (profile):** Vista protegida (@login_required) que permite al usuario ver y editar su propia información (UserUpdateForm, ProfileUpdateForm) y cambiar su contraseña (CustomPasswordChangeForm).

4.2. Flujo de Pacientes (pacientes)

- **CRUD:** Vistas estándar para listar, crear, ver, editar y eliminar pacientes, protegidas por roles (@personal_medico_required o @recepcionista_required). La creación de un paciente dispara la creación automática de un HistorialMedico vacío.
- **Validaciones:** PacienteForm valida formato de cédula (V/E + 7-8 dígitos), email único, fecha de nacimiento válida. DireccionForm y TelefonoForm validan sus campos respectivos (formato teléfono Vzla 11 dígitos).
- **Búsqueda:** Permite buscar pacientes por cédula, nombre o apellido.
- **Exportación:** Vistas (export_pacientes_pdf, export_pacientes_excel) generan archivos PDF o Excel del listado de pacientes, accesibles por personal médico.

- **AJAX Modals (crear_tipo_telefono_ajax):** Permite crear un TipoTelefono desde el formulario de Paciente sin recargar la página.

4.3. Flujo de Citas (citas)

- **CRUD Citas:** Vistas para listar, crear, ver, editar, eliminar citas, protegidas por roles. La creación/edición guarda una NotaCita de sistema.
- **CRUD Entidades Relacionadas:** Vistas separadas (protegidas por @admin_required) para gestionar EstadoCita, TipoCita, MotivoCita.
- **Validaciones (CitaForm):** Hora fin > hora inicio, fecha no pasada, no solapamiento de citas para el mismo paciente.
- **Funcionalidad Específica:** citas_hoy (vista para citas del día), cambiar_estado (vista para modificar estado), search (búsqueda).
- **Exportación:** Vistas (export_citas_pdf, export_citas_excel) generan reportes de citas.
- **AJAX Modals y Funcionalidad:**
    - Permite crear TipoCita, MotivoCita, EstadoCita modalmente.
    - Permite cambiar el estado de una cita (cambiar_estado_cita_ajax) directamente desde la lista mediante AJAX.

4.4. Flujo de Historiales (historiales)

- **CRUD:** Vistas para listar, crear, ver, editar, eliminar historiales, protegidas (@personal_medico_required). La creación solo permite seleccionar pacientes sin historial. La edición no permite cambiar el paciente.
- **Validaciones (HistorialMedicoForm):** Verifica que el paciente no tenga historial (en creación). Requiere al menos un campo clínico (Alergias, Enfermedades, Medicamentos).
- **Búsqueda:** Busca por paciente o contenido de los campos clínicos.
- **Exportación:** Vistas (export_historiales_pdf, export_historiales_excel) generan reportes de historiales.
- **AJAX Modals (crear_alergia_ajax, crear_enfermedad_ajax):** Permite crear Alergia y Enfermedad desde el formulario de Historial.

4.5. Flujo de Inventario (inventario)

- **CRUD Entidades:** Vistas CRUD estándar para Categoria, Proveedor, Medicamento, Inventario (existencias), protegidas por roles (@admin_required o @personal_medico_required).

- **Validaciones:** Se aplican las validaciones definidas en los formularios (CategoriaForm, ProveedorForm, MedicamentoForm, InventarioForm). La generación de código de Medicamento ocurre en el modelo.
- **Movimientos:**
    - listar_movimientos: Muestra el historial.
    - crear_entrada: (No implementada directamente, se asume que se hace creando un Inventario).
    - crear_salida: Vista específica para registrar salidas de stock usando MovimientoInventarioForm, valida que la cantidad no exceda el stock actual. Protegida (@personal_medico_required).
- **Stock (stock_medicamentos):** Muestra el stock actual calculado (Medicamento.stock_actual).
- **Exportación:** Vistas para exportar Categorías, Proveedores, Medicamentos, Inventario (existencias) y Stock a PDF/Excel.
- **AJAX Modals (crear_categoria_ajax, crear_proveedor_ajax, crear_medicamento_ajax):** Permiten crear estas entidades desde formularios relacionados (ej. crear Categoría desde el form de Medicamento).

4.6. Búsqueda Global (core)

- **search_all:** Vista que recibe un término de búsqueda y consulta los modelos Paciente, Cita e HistorialMedico (usando Q objects), devolviendo un contexto unificado a la plantilla search_results.html.

5. Dependencias y Entorno

5.1. Dependencias de Software

El archivo requirements.txt especifica las bibliotecas Python necesarias para ejecutar el proyecto:

- **Django==5.2.6:** El framework web principal utilizado para construir la aplicación.
- **psycopg2==2.9.10:** Adaptador de base de datos PostgreSQL para Python, permitiendo a Django interactuar con la base de datos PostgreSQL.
- **python-decouple==3.8:** Utilizado para separar parámetros de configuración (como claves secretas y detalles de la base de datos) del código fuente, leyéndolos desde variables de entorno o un archivo .env.
- **openpyxl==3.1.5:** Para la generación de archivos Excel (.xlsx).
- **xhtml2pdf==0.2.17:** Para la generación de archivos PDF desde plantillas HTML/CSS.
- **sqlparse==0.5.3:** Biblioteca no directamente usada por la aplicación, pero es una dependencia de Django utilizada internamente para formatear SQL.
- **asgiref==3.9.1:** Implementación de referencia ASGI, necesaria para el soporte asíncrono en Django.
- **tzdata==2025.2:** Provee información actualizada de zonas horarias, utilizada por Django para el manejo de fechas y horas conscientes de la zona horaria.

El frontend utiliza **Bootstrap 5.3.3**, **Bootstrap Icons 1.11.1**, y **jQuery 3.7.1** (incluidos localmente en static/) para la interfaz, componentes y funcionalidades AJAX/modales.

5.2. Configuración de Entorno

La configuración principal del proyecto se encuentra en sistema_medico/settings.py. Aspectos clave:

- **SECRET_KEY:** Clave secreta utilizada para la seguridad criptográfica de Django. Se carga desde el entorno usando python-decouple (config('SECRET_KEY')). Es crucial que esta clave sea única y se mantenga secreta en producción.
- **DEBUG:** Por defecto está configurado en True. Debe establecerse en False en un entorno de producción por razones de seguridad y rendimiento.
- **ALLOWED_HOSTS:** Lista vacía por defecto. En producción, debe configurarse con los dominios o direcciones IP desde los cuales se servirá la aplicación.
- **INSTALLED_APPS:** Define las aplicaciones que componen el proyecto: core, pacientes, citas, historiales, inventario, junto con las aplicaciones incorporadas de Django (admin, auth, contenttypes, etc.).
- **MIDDLEWARE:** Configura las capas de procesamiento de solicitudes/respuestas, incluyendo seguridad (SecurityMiddleware, CsrfViewMiddleware, XFrameOptionsMiddleware), sesiones (SessionMiddleware), autenticación (AuthenticationMiddleware), mensajes (MessageMiddleware), etc.
- **ROOT_URLCONF:** Establece sistema_medico.urls como el archivo principal de configuración de URLs.
- **TEMPLATES:** Configura el motor de plantillas de Django, especificando el directorio global templates/ y habilitando la búsqueda de plantillas dentro de cada aplicación (APP_DIRS: True).
- **DATABASES:** Configura la conexión a la base de datos. Utiliza python-decouple para cargar los parámetros (DB_NAME, DB_USER, DB_PASSWORD, DB_HOST, DB_PORT) desde el entorno, conectándose a una base de datos PostgreSQL.
- **AUTH_PASSWORD_VALIDATORS:** Define las reglas para la validación de contraseñas de usuario.
- **LOGIN_URL, LOGIN_REDIRECT_URL, LOGOUT_REDIRECT_URL:** Configuran las URLs para el flujo de autenticación, redirigiendo al dashboard después del login.
- **LANGUAGE_CODE:** Establecido en 'es' (Español).
- **TIME_ZONE:** Establecido en 'America/Caracas', asegurando que las fechas y horas se manejen según la zona horaria de Venezuela.
- **STATIC_URL:** Define la URL base para los archivos estáticos ('static/').

- **DEFAULT_AUTO_FIELD:** Configurado como BigAutoField, utilizando enteros de 64 bits para las claves primarias autoincrementales.

6. Guía de Instalación

Esta sección detalla los pasos recomendados para desplegar la Plataforma Web para el Control y Gestión del Servicio Médico UNEARTE en un entorno de servidor de producción. Se asume un entorno Linux (como Ubuntu/Debian) y el uso de PostgreSQL, Gunicorn y Nginx.

6.1. Prerrequisitos del Servidor

Asegúrese de que el servidor de producción tenga instalado lo siguiente:

- **Sistema Operativo:** Una distribución Linux actualizada *(ej. Ubuntu 22.04 LTS)*.
- **Python:** Versión 3.10 o superior.
- **pip:** Gestor de paquetes de Python.
- **virtualenv:** Herramienta para crear entornos virtuales de Python.
- **Git:** Sistema de control de versiones para clonar el repositorio.
- **PostgreSQL:** Servidor de base de datos (versión 12 o superior recomendada) y sus librerías de desarrollo (libpq-dev).
- **Nginx:** Servidor web/proxy inverso.
- **Gunicorn:** Servidor WSGI para Python (o una alternativa como uWSGI).
- **Supervisor (Opcional pero recomendado):** Para gestionar el proceso Gunicorn.
- **Certbot (Opcional pero recomendado):** Para configurar HTTPS con Let's Encrypt.

6.2. Configuración de la Base de Datos PostgreSQL

1. **Acceder a PostgreSQL:**
   Bash:
   sudo -u postgres psql

2. **Crear la Base de Datos:** (Reemplace nombre_db con el nombre deseado)
   SQL:
   CREATE DATABASE nombre_db WITH ENCODING 'UTF8';

3. **Crear un Usuario:** (Reemplace usuario_db y contraseña_segura con credenciales seguras)
   SQL:
   CREATE USER usuario_db WITH PASSWORD 'contraseña_segura';

4. **Configurar Privilegios:**

   SQL:
   ALTER ROLE usuario_db SET client_encoding TO 'utf8';

   ALTER ROLE usuario_db SET default_transaction_isolation TO 'read committed';

   ALTER ROLE usuario_db SET timezone TO 'America/Caracas'; -- Asegurar consistencia con settings.py

   GRANT ALL PRIVILEGES ON DATABASE nombre_db TO usuario_db;

5. **Salir de psql:**

   SQL:
   \q

6.3. Obtención y Configuración del Código Fuente

1. **Clonar el Repositorio:** (Navegue al directorio donde alojará la aplicación, ej. /opt/)

   Bash:
   cd /opt/

   git clone <URL_del_repositorio> unearte_medico

   cd unearte_medico

2. **Crear Entorno Virtual:**

   Bash:
   python3 -m venv venv

3. **Activar Entorno Virtual:**

   Bash:
   source venv/bin/activate

4. **Instalar Dependencias:**

Bash:
pip install -r requirements.txt

(*Esto instalará Django, psycopg2, python-decouple, openpyxl, xhtml2pdf y otras dependencias necesarias).*

5. **Crear Archivo de Variables de Entorno (.env):** En el directorio raíz del proyecto (unearte_medico/), cree un archivo llamado .env con el siguiente contenido, ajustando los valores según su configuración:

Ini, TOML:
```
# Archivo .env para producción

# Seguridad Django (¡Generar una nueva y mantenerla secreta!)
SECRET_KEY='su_clave_secreta_muy_larga_y_aleatoria_aqui'

# Configuración de Depuración (¡Obligatorio False en producción!)
DEBUG=False

# Hosts Permitidos (Dominio(s) y/o IP(s) del servidor)
ALLOWED_HOSTS=sudominio.com,www.sudominio.com,su_direccion_ip

# Configuración Base de Datos PostgreSQL
DB_NAME='nombre_db'
DB_USER='usuario_db'
DB_PASSWORD='contraseña_segura'
DB_HOST='localhost' # O la IP/host del servidor de BD si es externo
DB_PORT=5432

# Configuración Email (Ejemplo con SendGrid - Reemplazar con proveedor real)
# EMAIL_BACKEND='django.core.mail.backends.smtp.EmailBackend'
# EMAIL_HOST='smtp.sendgrid.net'
# EMAIL_PORT=587
# EMAIL_USE_TLS=True
# EMAIL_HOST_USER='apikey' # Literalmente 'apikey' para SendGrid
# EMAIL_HOST_PASSWORD='su_api_key_de_sendgrid'
# DEFAULT_FROM_EMAIL='noreply@sudominio.com'

# (Otras variables de entorno que la aplicación pueda requerir)
```

Importante: Asegúrese de que este archivo .env tenga permisos restrictivos (chmod 600 .env) para que solo el usuario que ejecuta la aplicación pueda leerlo.

6.4. Configuración de Django para Producción

1. **Aplicar Migraciones:**

   Bash:
   python manage.py migrate

2. **Recopilar Archivos Estáticos:**

   Bash:
   python manage.py collectstatic

   "*Esto copiará todos los archivos estáticos (CSS, JS, imágenes) al directorio especificado por STATIC_ROOT en settings.py (si no está definido, añadirlo; ej. STATIC_ROOT = BASE_DIR / 'staticfiles'). Nginx se configurará para servir archivos desde este directorio"*

3. **Crear un Superusuario Django:** (Para acceder al panel de administración /admin/)

   Bash:
   python manage.py createsuperuser

4. **Verificar Permisos:** Asegúrese de que el usuario que ejecutará Gunicorn tenga permisos de lectura/escritura sobre el directorio media/ (definido por MEDIA_ROOT en settings.py) si la aplicación permite la subida de archivos (como avatares).

6.5. Configuración del Servidor WSGI (Gunicorn)

1. **Probar Gunicorn Manualmente:** (Desde el directorio raíz del proyecto unearte_medico/ con el entorno virtual activado)

   Bash:
   gunicorn --workers 3 --bind unix:/tmp/unearte_medico.sock sistema_medico.wsgi:application

   "*Ajuste el número de --workers según los recursos del servidor (recomendado: 2 * num_cores + 1). Usar un socket Unix (unix:/tmp/...sock) es generalmente más rápido que un puerto TCP/IP para comunicación local con Nginx.*"

2. **Configurar un Servicio Systemd (Recomendado):** Cree un archivo /etc/systemd/system/gunicorn_unearte.service con el siguiente contenido (ajuste rutas y nombres de usuario/grupo):

   Ini, TOML:
   ```
   [Unit]
   Description=gunicorn daemon for UNEARTE Medico
   After=network.target

   [Service]
   User=usuario_sistema # Usuario bajo el cual correrá Gunicorn (NO root)
   Group=www-data     # Grupo (a menudo www-data para Nginx)
   WorkingDirectory=/opt/unearte_medico
   ExecStart=/opt/unearte_medico/venv/bin/gunicorn \
           --access-logfile - \
           --error-logfile - \
           --workers 3 \
           --bind unix:/run/gunicorn_unearte.sock \
           sistema_medico.wsgi:application

   [Install]
   WantedBy=multi-user.target
   ```

   Asegúrese de que el directorio /run/ exista y tenga permisos adecuados o ajuste la ruta del socket.

3. **Iniciar y Habilitar el Servicio:**

   Bash:
   sudo systemctl start gunicorn_unearte

   sudo systemctl enable gunicorn_unearte

   sudo systemctl status gunicorn_unearte # Verificar que esté corriendo

6.6. Configuración del Servidor Web (Nginx)

1. **Crear Archivo de Configuración de Nginx:** Cree un archivo /etc/nginx/sites-available/unearte_medico con el siguiente contenido (ajuste server_name, rutas y nombre del socket):

```
Nginx
server {
    listen 80;
        server_name sudominio.com www.sudominio.com; # Reemplazar con su dominio

    location = /favicon.ico { access_log off; log_not_found off; }

    # Servir archivos estáticos directamente
    location /static/ {
                root /opt/unearte_medico; # O la ruta donde ejecutó collectstatic (STATIC_ROOT)
    }

    # Servir archivos multimedia directamente
    location /media/ {
        root /opt/unearte_medico; # Ruta a la carpeta media (MEDIA_ROOT)
    }

    # Pasar solicitudes de la aplicación a Gunicorn
    location / {
        include proxy_params;
        proxy_pass http://unix:/run/gunicorn_unearte.sock;
    }

    # (Opcional: Añadir configuraciones de seguridad, logs, etc.)
```

}

2. **Habilitar el Sitio:**

   Bash:
   sudo ln -s /etc/nginx/sites-available/unearte_medico /etc/nginx/sites-enabled/

   sudo nginx -t # Verificar sintaxis

   sudo systemctl restart nginx

3. **Configurar HTTPS (Recomendado):** Utilice Certbot para obtener e instalar un certificado SSL/TLS gratuito de Let's Encrypt:

   Bash:
   sudo apt install certbot python3-certbot-nginx # Instalar Certbot

   sudo certbot --nginx -d sudominio.com -d www.sudominio.com # Obtener e instalar certificado

   “Certbot modificará automáticamente la configuración de Nginx para redirigir HTTP a HTTPS y usar el certificado.”

6.7. Configuración del Firewall

Asegúrese de que el firewall del servidor (ej. ufw) permita el tráfico en los puertos necesarios:

Bash:
sudo ufw allow 'Nginx Full' # Permite tráfico en puertos 80 (HTTP) y 443 (HTTPS)

sudo ufw enable

sudo ufw status

6.8. Verificación Final

Acceda a http://sudominio.com (o https://sudominio.com si configuró HTTPS) en su navegador. Debería ver la aplicación funcionando. Pruebe el login, la subida de archivos (si aplica) y las funcionalidades principales. Revise los logs de Gunicorn y Nginx (/var/log/nginx/) si encuentra errores.

7. Conclusión

La Plataforma Web para el Control y Gestión del Servicio Médico UNEARTE representa una solución tecnológica robusta y funcional, desarrollada con el framework Django y PostgreSQL, que aborda integralmente las necesidades identificadas en el servicio médico de la institución. El sistema centraliza la gestión de pacientes, citas, historiales médicos, inventario farmacéutico y usuarios, reemplazando procesos manuales ineficientes y propensos a errores.

Su arquitectura modular, basada en aplicaciones Django separadas por funcionalidad (core, pacientes, citas, historiales, inventario), promueve la mantenibilidad y escalabilidad futura. La implementación de un sistema de autenticación basado en roles, validaciones rigurosas en los formularios y el uso de transacciones atómicas en las vistas garantizan la integridad y seguridad de los datos. La interfaz de usuario, construida con Bootstrap y mejorada con interacciones dinámicas, ofrece una experiencia moderna y accesible.

Este sistema no solo optimiza la operatividad interna del servicio médico mediante la digitalización y la inclusión de herramientas como la generación de reportes y búsqueda avanzada, sino que también mejora la calidad y accesibilidad de la atención para la comunidad universitaria de UNEARTE (estudiantes, docentes, personal administrativo). Como proyecto de servicio comunitario evaluado por la UNETI, cumple con los objetivos académicos y sociales propuestos, entregando una herramienta valiosa y sostenible.

Esta documentación técnica sirve como guía para el equipo de UNEARTE encargado de su mantenimiento, proporcionando una visión detallada de su estructura, lógica y configuración, sentando las bases para futuras mejoras y adaptaciones.