**Análisis Detallado del Plan de Desarrollo Frontend**

Vamos a examinar cada fase del plan de desarrollo para asegurarnos que cubre todas las necesidades del proyecto Masclet Imperi y se alinea con la visión que tienes. Analizaré las ventajas, desafíos y consideraciones específicas de cada etapa.

**Fase 1: Configuración y Estructura Base (1 semana)**

**Actividades detalladas:**

1. **Inicialización del proyecto Astro**:
   * Crear un nuevo proyecto usando **npm create astro@latest**
   * Configurar TypeScript desde el inicio
   * Estructurar carpetas según el plan definido
2. **Integración de Tailwind CSS 4.0**:
   * Instalar como dependencia de desarrollo
   * Configurar el archivo **tailwind.config.mjs** con:
     + Colores personalizados para el sector ganadero
     + Extensiones de tema para estados específicos (OK, DEF)
     + Configuración de fuentes y tamaños
3. **Desarrollo del layout principal**:
   * **Navbar.astro** con detección de rol de usuario
   * **Sidebar.astro** configurable según permisos
   * **Footer.astro** con información de versión y contacto
   * **LayoutDefault.astro** como plantilla principal
4. **Configuración de servicios API**:
   * Crear cliente Axios con interceptores para:
     + Manejo de tokens JWT
     + Manejo centralizado de errores
     + Transformación de datos

**Desafíos potenciales:**

* Curva de aprendizaje inicial de Astro (diferente a React puro)
* Configuración correcta para permitir componentes React dentro de Astro
* Asegurar la reutilización de componentes entre ambos frameworks

**Resultados esperados:**

* Estructura sólida del proyecto lista para crecer
* Sistema de diseño con Tailwind establecido
* Comunicación con la API configurada
* Esquema de navegación principal funcional

**Fase 2: Dashboard (2 semanas)**

**Actividades detalladas:**

1. **Implementación de la página principal**:
   * Diseño responsive con grid de Tailwind
   * Secciones claramente definidas por tipo de contenido
2. **Desarrollo de componentes estadísticos**:
   * **StatsCard.tsx** para KPIs principales:
     + Total animales por explotación
     + Distribución por género (M/F)
     + Estados de amamantamiento (alletar)
3. **Gráficos interactivos**:
   * **GenderChart.tsx**: Distribución por género
   * **StatusChart.tsx**: Estados OK/DEF
   * **AlletarChart.tsx**: Estados de amamantamiento
   * **PartosTimeline.tsx**: Evolución temporal de partos
4. **Integración con backend**:
   * **dashboardService.ts** para consumir **/api/dashboard/stats**
   * Gestión de filtros por explotación, fecha, etc.
   * Cacheo inteligente para mejorar rendimiento

**Desafíos potenciales:**

* Optimización del rendimiento con múltiples gráficos en pantalla
* Visualización apropiada en dispositivos móviles
* Manejo adecuado de estados de carga y error

**Resultados esperados:**

* Dashboard completamente funcional
* Datos actualizados en tiempo real
* Capacidad de filtrado por explotaciones
* Vista adaptable a cualquier dispositivo

**Fase 3: Gestión de Animales (2 semanas)**

**Actividades detalladas:**

1. **Listado de animales**:
   * Tabla interactiva con:
     + Paginación eficiente
     + Ordenamiento por columnas
     + Filtros avanzados (explotación, género, estado)
   * Vista en modo tarjeta para móviles
2. **Vista detallada de animal**:
   * Información completa con tabs para:
     + Datos básicos
     + Historial de partos
     + Relaciones (padre/madre)
     + Cuadra/ubicación
3. **Formularios**:
   * Creación con validaciones en tiempo real
   * Edición con carga previa de datos
   * Manejo de campos obligatorios según reglas de negocio
   * Validaciones específicas (num\_serie formato ES+números)
4. **Funcionalidades especiales**:
   * Gestión de estado "alletar"
   * Cambio de estado (OK/DEF)
   * Asignación de cuadra

**Desafíos potenciales:**

* Validaciones complejas según reglas de negocio
* UX para la introducción de datos específicos (fechas, códigos)
* Rendimiento con grandes listas de animales

**Resultados esperados:**

* CRUD completo de animales
* Sistema de filtrado avanzado
* Validaciones robustas
* Interfaz intuitiva para ganaderos

**Fase 4: Gestión de Partos (2 semanas)**

**Actividades detalladas:**

1. **Registro de partos**:
   * Formulario especializado para:
     + Selección de madre (solo hembras)
     + Fecha del parto (validación de formato DD/MM/YYYY)
     + Género de la cría (M/F/esforrada)
     + Estado de la cría (OK/DEF)
2. **Historial por animal**:
   * Vista cronológica de partos
   * Filtrado por fecha
   * Estadísticas de éxito/fracaso
3. **Integración con módulo de animales**:
   * Actualización automática del estado "alletar"
   * Vínculo directo a creación de nuevo animal por parto
   * Relación madre-cría automática
4. **Validaciones de negocio**:
   * Solo hembras pueden tener partos
   * Control de plazos entre partos
   * Validación de estados coherentes

**Desafíos potenciales:**

* Complejidad de la lógica de negocio
* Mantener la integridad referencial
* Experiencia de usuario para flujos complejos

**Resultados esperados:**

* Sistema completo de gestión de partos
* Automatización de relaciones entre registros
* Visualización clara del historial
* Validaciones que garanticen datos correctos

**Fase 5: Sistema de Importación (1 semana)**

**Actividades detalladas:**

1. **Interfaz de carga**:
   * Drag & drop para archivos CSV
   * Validación de formato antes de envío
   * Progreso visual durante la carga
2. **Previsualización de datos**:
   * Tabla con primeras 10-20 filas
   * Identificación visual de posibles errores
   * Mapeo de columnas configurable
3. **Proceso de importación**:
   * Envío por chunks para archivos grandes
   * Barra de progreso detallada
   * Cancelación segura
4. **Reporte de resultados**:
   * Resumen de registros procesados
   * Detalle de errores encontrados
   * Opciones de corrección o reintento

**Desafíos potenciales:**

* Manejo de archivos grandes
* Interfaz intuitiva para no técnicos
* Feedback claro sobre errores en datos

**Resultados esperados:**

* Sistema robusto de importación
* Validación previa para evitar errores
* Experiencia de usuario simple
* Reportes detallados del proceso

**Fase 6: Autenticación y Permisos (1 semana)**

**Actividades detalladas:**

1. **Sistema de login**:
   * Formulario seguro con validaciones
   * Manejo de tokens JWT
   * Persistencia de sesión configurable
   * Recuperación de contraseña
2. **Gestión de permisos**:
   * Implementación de guard routes basado en roles
   * Mostrar/ocultar funcionalidades según permisos
   * Feedback visual para acciones no permitidas
3. **Panel de administración**:
   * Gestión de usuarios (solo ADMIN)
   * Asignación de roles y explotaciones
   * Auditoría de acciones
4. **Seguridad**:
   * Protección contra CSRF
   * Expiración y renovación de tokens
   * Logout automático por inactividad

**Desafíos potenciales:**

* Seguridad robusta sin complicar la UX
* Gestión correcta de permisos en UI
* Mantener rendimiento con validaciones constantes

**Resultados esperados:**

* Sistema de autenticación seguro
* Control de acceso granular
* Gestión de usuarios para administradores
* Protección de rutas y funcionalidades sensibles

**Fase 7: Refinamiento y Optimización (1 semana)**

**Actividades detalladas:**

1. **Pruebas de usabilidad**:
   * Sesiones con usuarios finales
   * Ajustes basados en feedback
   * Mejora de flujos complejos
2. **Optimización de rendimiento**:
   * Code splitting para reducir bundles
   * Lazy loading de componentes pesados
   * Optimización de assets (imágenes, iconos)
   * Auditoría con Lighthouse
3. **Responsive design**:
   * Verificación en múltiples dispositivos
   * Mejora de interacciones táctiles
   * Refinamiento de layouts móviles
4. **Accesibilidad**:
   * Revisión de contraste
   * Navegación por teclado
   * Etiquetas ARIA apropiadas
   * Compatibilidad con lectores de pantalla

**Desafíos potenciales:**

* Balancear optimizaciones técnicas con experiencia de usuario
* Mantener coherencia visual en todos los dispositivos
* Implementar accesibilidad sin comprometer diseño

**Resultados esperados:**

* Aplicación pulida y profesional
* Rendimiento optimizado
* Experiencia consistente en todos los dispositivos
* Niveles adecuados de accesibilidad

**Visión General y Próximos Pasos**

Este análisis detallado muestra que el plan cubre todos los aspectos críticos del desarrollo frontend para Masclet Imperi. Las 7 fases están diseñadas para construir progresivamente la aplicación, empezando por los cimientos y avanzando hacia funcionalidades más complejas.

**Alternativas de diseño Tailwind CSS**

Basándonos en tu interés por explorar diseños alternativos de Tailwind, aquí te presento opciones inspiradas en los ejemplos que mencionaste del showcase de Tailwind:

**1. Estilo "Conference Website"**

Un diseño más orientado a eventos y presentaciones. Para Masclet Imperi podría traducirse en:

* Secciones de dashboard con tarjetas más visuales y grandes
* Uso de fondos con patrones sutiles o gradientes
* Banners destacados para anuncios importantes (próximos partos, alertas)
* Mayor espaciado entre elementos

**2. Estilo "Candy Code" (Diseño moderno y colorido)**

* Paleta de colores más vibrante y contrastante
* Bordes redondeados más pronunciados
* Iconos con más presencia visual y animaciones
* Uso de sombras y efectos de hover más llamativos

**3. Estilo "Lemon Squeezy" (Minimalista profesional)**

Este estilo sería ideal para una aplicación de gestión:

* Interfaz limpia con mucho espacio en blanco
* Colores pastel suaves con acentos de color para las acciones
* Tipografía moderna y bien espaciada
* Tarjetas con bordes suaves y sombras sutiles
* Tablas con diseño minimalista pero muy legible

**4. Estilo "Personal Website/Spotlight"**

Adaptado para una aplicación de gestión ganadera:

* Páginas de perfil para cada animal con diseño tipo "spotlight"
* Imágenes destacadas y estadísticas importantes
* Historial de cada animal presentado como una línea de tiempo visual
* Navegación lateral con efecto de resaltado para la sección actual

**5. Estilo "Primer" (Inspirado en GitHub)**

Perfecto para una aplicación con mucha información y datos:

* Diseño más denso pero muy organizado
* Sistema de colores para codificar estados (OK/DEF)
* Uso de pestañas para organizar información relacionada
* Componentes compactos pero altamente funcionales
* Tablas con ordenación y filtrado avanzado

**Pasos para implementar un nuevo estilo**

Si quieres adoptar alguno de estos estilos, podríamos:

1. **Actualizar el archivo Tailwind**:
   * Modificar la paleta de colores
   * Personalizar los componentes base
   * Definir nuevos elementos UI
2. **Crear componentes UI base**:
   * Botones, tarjetas, tablas con el nuevo estilo
   * Implementar sistema de iconos consistente
3. **Aplicar gradualmente el nuevo estilo**:
   * Empezar por el dashboard como página principal
   * Adaptar progresivamente el resto de páginas
4. **Desarrollar una biblioteca de componentes**:
   * Documentar componentes reutilizables
   * Crear versiones para diferentes tamaños y estados

**Plan de Desarrollo para la Implementación de la API en el Frontend**

**1. Estructura de Autenticación y Autorización**

**Páginas y Componentes**

* **Página de Login**: Implementar una página de inicio de sesión antes de acceder a cualquier otra parte de la aplicación
* **Gestión de Tokens**: Asegurar que el token JWT se almacene y gestione correctamente
* **Protección de Rutas**: Implementar protección de rutas basada en roles (admin, manager, editor, user)
* **Componente de Logout**: Añadir funcionalidad para cerrar sesión

**Servicios**

* **authService**: Completar la implementación de login, logout, verificación de tokens
* **Interceptores**: Configurar interceptores para añadir tokens a todas las peticiones

**2. Estructura de Navegación y Layout**

* **Layout Principal**: Crear un layout común con sidebar, header y área de contenido
* **Menú de Navegación**: Implementar un menú dinámico basado en los permisos del usuario
* **Breadcrumbs**: Añadir navegación por migas de pan para mejorar la UX

**3. Implementación de Módulos Principales**

**Módulo de Animales**

* **Listado de Animales**: Vista con filtros, paginación y ordenación
* **Detalle de Animal**: Vista detallada con información completa
* **Formulario de Creación/Edición**: Implementar formularios con validaciones
* **Eliminación**: Confirmación y gestión de eliminación

**Módulo de Partos**

* **Listado de Partos**: Vista con filtros y relación con animales
* **Detalle de Parto**: Vista detallada con información completa
* **Formulario de Creación/Edición**: Implementar formularios con validaciones
* **Eliminación**: Confirmación y gestión de eliminación

**Módulo de Explotaciones**

* **Listado de Explotaciones**: Vista con filtros
* **Detalle de Explotación**: Vista detallada con información completa
* **Formulario de Creación/Edición**: Implementar formularios con validaciones

**Módulo de Importación**

* **Importación CSV**: Mejorar la funcionalidad actual
* **Historial de Importaciones**: Implementar vista de historial
* **Plantillas**: Gestión de plantillas para descarga

**Módulo de Dashboard**

* **Resumen General**: Estadísticas principales
* **Gráficos**: Implementar visualizaciones de datos
* **Filtros**: Añadir filtros para personalizar la vista

**Módulo de Usuarios (Solo Admin)**

* **Gestión de Usuarios**: CRUD completo
* **Asignación de Roles**: Interfaz para cambiar roles
* **Reseteo de Contraseñas**: Funcionalidad para administradores

**4. Integración de Servicios API**

Para cada módulo, implementar los servicios correspondientes:

* **animalService**: Completar CRUD y operaciones específicas
* **partoService**: Completar CRUD y operaciones específicas
* **explotacionService**: Completar CRUD y operaciones específicas
* **importService**: Mejorar la gestión de importaciones
* **dashboardService**: Implementar obtención de estadísticas
* **userService**: Implementar gestión de usuarios

**5. Gestión de Estado y Datos**

* **Estado Global**: Evaluar necesidad de implementar gestión de estado (Context API, Redux)
* **Caché**: Implementar estrategias de caché para mejorar rendimiento
* **Manejo de Errores**: Sistema centralizado de manejo de errores

**6. Testing**

* **Tests Unitarios**: Implementar para componentes clave
* **Tests de Integración**: Probar flujos completos
* **Tests E2E**: Automatizar pruebas de flujos críticos

**7. Mejoras de UX/UI**

* **Feedback Visual**: Mejorar indicadores de carga, éxito y error
* **Formularios**: Mejorar validaciones y experiencia de usuario
* **Responsive Design**: Asegurar que la aplicación funcione en diferentes dispositivos
* **Accesibilidad**: Implementar mejoras de accesibilidad

**8. Despliegue y CI/CD**

* **Configuración de Entornos**: Desarrollo, Staging, Producción
* **Variables de Entorno**: Gestión adecuada de configuraciones
* **Pipeline de CI/CD**: Automatizar pruebas y despliegue

**Próximos Pasos Inmediatos**

1. **Implementar página de login y protección de rutas**
2. **Completar el servicio de autenticación**
3. **Crear layout principal con navegación basada en roles**
4. **Implementar la primera versión funcional del listado de animales**

las reglas del programa base son claras:  
  
4 roles, ADMINISTRADOR, GERENTE, EDITOR, USUARIOS  
administrador tiene permiso a todo que básicamente es:  
CONSULTAR FICHAS DE ANIMALES (acceso para administrador y Gerente y editor y usuario)  
todos los datos y cada uno con su icono según sean toros, vacas sin amamantar y vaca amamantando (con un ternero o dos) y fallecidos  
CONSULTAR LISTADOS DE EXPLOTACIONES (acceso para administrador y Gerente y editor y usuario)  
aquí hay que ver que campos e iconos quiero que se visualicen con el Listado de explotación (que no serán todos) todos los animales pertenecientes a esa explotación (cada uno con su icono según sean toros, vacas sin amamantar y vaca amamantando (con un ternero o dos) y fallecidos)  
  
ACTUALIZAR FICHAS DE ANIMALES (acceso para administrador y Gerente y editor)  
en 2 grupos principales:  
DATOS GENERALES:(esto son cambios que se van a ir haciendo pero cuando los datos ya sean fiables cambiaran poco)  
nombre  
explotación  
genero  
padre  
madre  
cuadra  
COD  
numero de serie  
DOB  
aquí también se listaran los datos de todos los partos que ha tenido cada vaca, con sus campos de numero de parto, fecha del parto, genero del ternero y estado del ternero  
CAMBIOS HABITUALES (estos son los que el gerente actualizara a diario)  
estado: 2 estados OK o DEF  
amamantando: 3 estados (NO, 1 ternero, 2 terneros) solo para vacas (genero femenino)  
datos de nuevo parto (fecha del parto, genero del ternero y estado del ternero)  
  
NUEVA FICHA (acceso para administrador y Gerente)  
aquí se crearan fichas nuevas cuando un ternero se haga mayor y pase a formar parte de la explotación N o se compre un toro o una vaca (esta ficha ya esta añadida en el frontend con el icono de + nuevo animal  
  
IMPORTACIONES (acceso a administrador)  
traer un csv con paquetes de datos nuevos, la importación no puede borrar datos existentes, debe sumarlos y si hay algún nombre duplicado debe el proceso de importación avisar de la incompatibilidad y grabar el nombre añadiendo algún carácter par que en la importación se mantenga todo  
  
GESTION DE USUARIOS (acceso para administrador y Gerente)  
creación de usuarios y perfiles  
actualización de datos de usuarios y perfiles  
borrado de usuarios y perfiles  
solo puede haber un administrador y no se puede borrar, y solo puede haber un gerente  
  
se ha creado un nuevo campo de BACKUP (muy bueno) acceso solo para administrador  
  
los partos los hemos puesto como un campo aparte pero en principio no los quiero separados (de momento que se queden así, por si acaso, pero trabajaremos en lo de arriba y ya veremos esto.  
  
principios que están en contexto pero que merece la pena recordar, los terneros solo están contabilizados por los partos de sus madres vacas, no tienen ficha propiamente dicha, cuando crecen o bien se venden (y desaparecen) o bien pasan a formar parte de una explotación y entonces si se les hace ficha

**Orden estratégico para implementar la conexión API**

1. **Configuración de entorno (.env)**:
   * Verificar que ambos entornos (frontend y backend) tengan las variables necesarias para conectarse
2. **Servicio API base**:
   * Ya hemos trabajado en **apiService.ts** como capa de abstracción, lo cual es excelente
3. **Autenticación**:
   * **authService.ts** - Es crítico porque todo el sistema depende de la autenticación funcionando correctamente
   * Recordando que la ruta correcta es **/api/v1/auth/login** (no **/auth/login**)
4. **Servicios centrales de entidades** (en orden de prioridad):
   * **animalService.ts** - Es el núcleo de la aplicación
   * **explotacioService.ts** - Necesario para contextualizar los animales
   * **partService.ts** - Relacionado con eventos de animales
   * **dashboardService.ts** - Depende de los otros servicios
   * **importService.ts** - Funcionalidad avanzada que puede esperar
5. **Interfaces y tipos**:
   * Verificar que la estructura de datos y tipos en el frontend coincida con el backend

**La arquitectura ideal para tu caso**

1. **Frontend + Backend separados**:
   * Frontend: Desplegado en Vercel o Netlify (servicio gratuito/económico)
   * Backend: Desplegado en un servidor VPS económico (DigitalOcean, Hetzner)
2. **Para el problema de autenticación actual**:
   * La solución de API Route/proxy es perfecta porque:
     + Funciona tanto en desarrollo como en producción
     + No necesita configuración adicional al desplegar
     + Evita completamente los problemas CORS
3. **Optimizaciones para usuarios móviles**:
   * Almacenamiento en caché para funcionamiento offline
   * Diseño responsivo prioritario
   * Carga progresiva para conexiones lentas

**Pasos concretos para hacer la aplicación autónoma y robusta**

1. **Cambiar configuración Astro** como hemos hecho (**output: 'server'**)
2. **Implementar proxy de autenticación** como estamos haciendo
3. **Añadir sistema de caché** para datos frecuentes
4. **Configurar monitorización simple** (correos automáticos si hay error)

Una vez desplegado, tus amigos ganaderos tendrán una aplicación robusta que raramente necesitará tu intervención, y cuando estén en el campo con conexión limitada seguirá funcionando gracias al enfoque que estamos implementando.

todo eso cruzalo con esto: 4 roles, ADMINISTRADOR, GERENTE, EDITOR, USUARIOS administrador tiene permiso a todo que básicamente es: 1.CONSULTAR FICHAS DE ANIMALES (acceso para administrador y Gerente y editor y usuario) todos los datos y cada uno con su icono según sean toros, vacas sin amamantar y vaca amamantando (con un ternero o dos) y fallecidos 2.CONSULTAR LISTADOS DE EXPLOTACIONES (acceso para administrador y Gerente y editor y usuario) aquí hay que ver que campos e iconos quiero que se visualicen con el Listado de explotación (que no serán todos) todos los animales pertenecientes a esa explotación (cada uno con su icono según sean toros, vacas sin amamantar y vaca amamantando (con un ternero o dos) y fallecidos) 3.ACTUALIZAR FICHAS DE ANIMALES (acceso para administrador y Gerente y editor) en 2 grupos principales: 3.1.DATOS GENERALES:(esto son cambios que se van a ir haciendo pero cuando los datos ya sean fiables cambiaran poco) nombre explotación genero padre madre cuadra COD numero de serie DOB aquí también se listaran los datos de todos los partos que ha tenido cada vaca, con sus campos de numero de parto, fecha del parto, genero del ternero y estado del ternero 3.2.CAMBIOS HABITUALES (estos son los que el gerente actualizara a diario) estado: 2 estados OK o DEF amamantando: 3 estados (NO, 1 ternero, 2 terneros) solo para vacas (genero femenino) datos de nuevo parto (fecha del parto, genero del ternero y estado del ternero) 4.NUEVA FICHA (acceso para administrador y Gerente) aquí se crearan fichas nuevas cuando un ternero se haga mayor y pase a formar parte de la explotación N o se compre un toro o una vaca (esta ficha ya esta añadida en el frontend con el icono de + nuevo animal 5.IMPORTACIONES (acceso a administrador) traer un csv con paquetes de datos nuevos, la importación no puede borrar datos existentes, debe sumarlos y si hay algún nombre duplicado debe el proceso de importación avisar de la incompatibilidad y grabar el nombre añadiendo algún carácter par que en la importación se mantenga todo 6.GESTION DE USUARIOS (acceso para administrador y Gerente) creación de usuarios y perfiles actualización de datos de usuarios y perfiles borrado de usuarios y perfiles solo puede haber un administrador y no se puede borrar, y solo puede haber un gerente 7.se ha creado un nuevo campo de BACKUP (muy bueno) acceso solo para administrador los partos los hemos puesto como un campo aparte pero en principio no los quiero separados de momento con esto: Router Structure: Authentication: /api/v1/auth/ Explotaciones (Farms): /api/v1/explotacions/ Animals: /api/v1/animals/ Partos (Births) - nested within animals: /api/v1/animals/{animal\_id}/parts/ Partos (standalone): /api/v1/partos/ Dashboard: /api/v1/dashboard/ Imports: /api/v1/imports/ Authentication Endpoints: Login: POST /api/v1/auth/login Register user: POST /api/v1/auth/register Get current user info: GET /api/v1/auth/me Refresh token: POST /api/v1/auth/refresh Get users list: GET /api/v1/auth/users Delete user: DELETE /api/v1/auth/users/{user\_id} Change password: PUT /api/v1/auth/users/{user\_id}/password Animal Endpoints: Create animal: POST /api/v1/animals/ Get animal details: GET /api/v1/animals/{animal\_id} List animals: GET /api/v1/animals/ (with query parameters) Update animal: PUT /api/v1/animals/{animal\_id} Delete animal: DELETE /api/v1/animals/{animal\_id} I should also consider the user's requirements for role-based access control: ADMINISTRADOR: Full access to all features GERENTE: Access to animal management, farm listings, and user management (except admin) EDITOR: Access to animal viewing and updating USUARIOS: Access to viewing animals and farms only 0. Dashboard Principal (Página de Inicio tras Login) Los endpoints del dashboard ya están implementados en el backend y proporcionan estadísticas detalladas: /api/v1/dashboard/stats: Estadísticas completas con filtros opcionales /api/v1/dashboard/explotacions/{id}: Estadísticas específicas por explotación /api/v1/dashboard/partos: Análisis detallado de partos /api/v1/dashboard/combined: Vista consolidada de todas las estadísticas /api/v1/dashboard/activity: Actividad reciente Principios Básicos del Programa Masclet Imperi Basado en la documentación de contexto del proyecto, estos son los principios fundamentales: Datos y Estructura Básica Animales: Identificados por nombre, explotación, género y estado Géneros: Machos (M) y Hembras (F) Estados: Activo (OK) y Fallecido (DEF) Amamantamiento (Alletar): Solo para vacas, con estados (0, 1, 2 terneros) Partos: Registro histórico permanente (fecha, género y estado del ternero) Roles y Permisos Administrador: Acceso total (único en el sistema) Gerente: Gestión de explotaciones, creación y modificación de fichas Editor: Actualización de datos habituales (estados, amamantamiento, partos) Usuario: Solo consulta de datos Filosofía de Uso y Diseño Contexto Rural: Interfaz adaptada para ganaderos trabajando en campo Dispositivos: Principalmente móviles y tablets, en exteriores Conectividad: Debe funcionar con conexión inestable o intermitente (sera una excepcion no tener internet) Usabilidad: Controles grandes, tolerancia a errores, interfaces simples Características Técnicas Offline: Capacidad de funcionamiento sin conexión constante (esto hay que ver riesgos y fortalezas) Sincronización: Sistema para actualizar cuando hay conexión Validación Flexible: Prioriza el registro sobre la validación estricta Iconografía: Sistema visual para identificar tipos de animales Módulos Principales Dashboard: Estadísticas en tiempo real de explotaciones y animales Fichas de Animales: Consulta detallada con historial de partos Explotaciones: Listado y gestión por ubicación Registro de Partos: Seguimiento de reproducción (siempre asociada a las vacas que tienen los partos, los terneros no tienen entidad propia) Importaciones: Carga masiva de datos CSV Gestión de Usuarios: Control de accesos por rol Backup: Sistema de respaldo de datos