



LA RESERVA - DESARROLLO WEB

PARTE C-1: Estructura del Proyecto

Versión: 1.0

Fecha: Octubre 2025

Tiempo de lectura: 12 minutos





CONTENIDO

1. Estructura Completa del Proyecto
2. Explicación de Directorios
3. Sistema de Rutas de Astro
4. API Endpoints


1. ESTRUCTURA COMPLETA DEL PROYECTO

```
la-reserva/
├── public/                                # Archivos estáticos
│   ├── favicon.svg
│   ├── logo.svg
│   ├── robots.txt
│   ├── sitemap.xml
│   └── images/
│       ├── events/                       # Fotos de eventos
│       ├── cocktails/                   # Fotos de cócteles
│       ├── team/                         # Fotos del equipo
│       └── brand/                       # Assets de marca
├── src/
│   ├── assets/                           # Assets procesados por Astro
│   │   ├── images/
│   │   └── icons/
│   └── components/                       # Componentes reutilizables
```

└─  layout/	# Componentes de layout
└─ Header.astro	
└─ Footer.astro	
└─ Navbar.astro	
└─ MobileMenu.astro	
└─  ui/	# Componentes UI base
└─ Button.astro	
└─ Card.astro	
└─ Modal.tsx	
└─ Input.astro	
└─ Badge.astro	
└─ Toast.tsx	
└─ Loading.astro	
└─  sections/	# Secciones de páginas
└─ Hero.astro	
└─ Services.astro	
└─ Portfolio.astro	
└─ Testimonials.astro	
└─ CTA.astro	
└─ Contact.astro	
└─  forms/	# Formularios
└─ QuoteForm.tsx	
└─ ContactForm.tsx	
└─ NewsletterForm.tsx	
└─  admin/	# Componentes del panel admin
└─ Dashboard.tsx	
└─ EventCalendar.tsx	
└─ QuotesList.tsx	
└─ EventsList.tsx	
└─ ClientsList.tsx	

└─  layouts/	# Layouts de página
└─ BaseLayout.astro	# Layout base (SEO, meta)
└─ PageLayout.astro	# Layout de páginas públicas
└─ AdminLayout.astro	# Layout del panel admin
└─  pages/	# Rutas del sitio (file-based routing)
└─ index.astro	# Home (/)
└─ servicios.astro	# Servicios (/servicios)
└─ paquetes.astro	# Paquetes (/paquetes)

```
├── portafolio.astro # Portafolio (/portafolio)
├── nosotros.astro # Sobre nosotros (/nosotros)
├── blog/
│   ├── index.astro # Blog index (/blog)
│   └── [slug].astro # Post individual (/blog/[slug])
├── contacto.astro # Contacto (/contacto)
├── cotizacion.astro # Formulario cotización (/cotizac
├──
├── 📁 admin/ # Panel administrativo
│   ├── index.astro # Dashboard (/admin)
│   ├── login.astro # Login (/admin/login)
│   ├── eventos.astro # Gestión eventos (/admin/eventos)
│   ├── cotizaciones.astro # Gestión cotizaciones (/admin/cc
│   ├── clientes.astro # Gestión clientes (/admin/client
│   ├── contenido.astro # Gestión contenido (/admin/conte
│   └── configuracion.astro # Configuración (/admin/configura
├──
├── 📁 api/ # API endpoints (opcional)
│   ├── quotes.ts # POST /api/quotes
│   └── contact.ts # POST /api/contact
├──
├── 📁 lib/ # Librerías y configuraciones
│   ├── supabase.ts # Cliente Supabase
│   ├── resend.ts # Cliente Resend (emails)
│   └── whatsapp.ts # Helpers WhatsApp
├──
├── 📁 utils/ # Funciones de utilidad
│   ├── utils.ts # Utilidades generales (cn, etc.)
│   ├── constants.ts # Constantes del proyecto
│   ├── validators.ts # Validadores con Zod
│   └── formatters.ts # Formateo de datos
├──
├── 📁 types/ # Tipos TypeScript
│   └── index.ts # Tipos globales
├──
├── 📁 styles/ # Estilos globales
│   └── global.css # Estilos CSS globales
├──
├── 📁 content/ # Collections de contenido (opcio
│   ├── config.ts # Configuración de collections
│   └── 📁 blog/ # Posts del blog (Markdown)
│       ├── primer-post.md
│       └── segundo-post.md
```

└─ env.d.ts	# Tipos de entorno
└─  supabase/	# Configuración de Supabase
└─ migrations/	# Migraciones SQL
└─ functions/	# Edge Functions
└─ seed.sql	# Datos iniciales
└─ .env	# Variables de entorno (NO SUBIR)
└─ .env.example	# Ejemplo de variables
└─ .gitignore	
└─ .prettierrc	
└─ .prettierignore	
└─ astro.config.mjs	
└─ package.json	
└─ pnpm-lock.yaml	
└─ tailwind.config.mjs	
└─ tsconfig.json	
└─ README.md	

2. EXPLICACIÓN DE DIRECTORIOS

public/

Propósito: Archivos estáticos que se sirven directamente sin procesamiento.

Contenido:

- `favicon.svg` - Icono del sitio
- `logo.svg` - Logo en SVG
- `robots.txt` - Instrucciones para crawlers
- `sitemap.xml` - Mapa del sitio para SEO
- `images/` - Imágenes optimizadas manualmente

Regla: Estos archivos se sirven desde la raíz del dominio.

- `public/logo.svg` → `https://lareserva.pe/logo.svg`

Ejemplo de robots.txt:

```
# public/robots.txt
User-agent: *
Allow: /
```

Sitemap: <https://lareserva.pe/sitemap.xml>

src/assets/

Propósito: Assets que serán procesados y optimizados por Astro.

Ventajas:

- Optimización automática de imágenes
- Hashing de archivos para cache
- Import directo en componentes

Ejemplo de uso:

```
---  
import logo from '@assets/images/logo.png';  
import { Image } from 'astro:assets';  
---  
  
<Image  
  src={logo}  
  alt="La Reserva"  
  width={200}  
  height={100}  
  format="webp"  
>
```

src/components/

Propósito: Componentes reutilizables de la UI.

Organización:

- **layout/** - Header, Footer, Navbar (estructura principal)
- **ui/** - Componentes base (Button, Card, Modal)
- **sections/** - Secciones grandes (Hero, Services)
- **forms/** - Formularios (React con validación)
- **admin/** - Componentes del panel (React)

Convención de nombres:

- `.astro` para componentes estáticos
- `.tsx` para componentes con interactividad (React)

Ejemplo de estructura:

```

components/
├── layout/
│   ├── Header.astro           # Componente estático
│   └── MobileMenu.tsx        # Componente interactivo (React)
├── ui/
│   ├── Button.astro          # Componente visual simple
│   └── Modal.tsx             # Componente con estado (React)
└── forms/
    └── QuoteForm.tsx         # Formulario complejo (React)

```

`src/layouts/`

Propósito: Plantillas de página que envuelven contenido.

Layouts principales:

1. **BaseLayout.astro** - Layout mínimo con:

- HTML base
- Meta tags SEO
- Google Fonts
- Scripts globales

2. **PageLayout.astro** - Layout de páginas públicas:

- Extiende BaseLayout
- Incluye Header + Footer
- Estilos específicos de página

3. **AdminLayout.astro** - Layout del panel admin:

- Sidebar de navegación
- Sin Header/Footer público
- Protegido con autenticación

Jerarquía:

`BaseLayout.astro` (base HTML + SEO)

↓

PageLayout.astro (Header + Footer)



Contenido de página (slot)

src/pages/

Propósito: Sistema de rutas file-based de Astro.

Reglas:

- Cada archivo `.astro` o `.ts` = una ruta
- `index.astro` = ruta raíz del directorio
- `[param].astro` = ruta dinámica
- Carpetas = segmentos de URL

Ejemplos:

<code>src/pages/index.astro</code>	→ <code>/</code>
<code>src/pages/servicios.astro</code>	→ <code>/servicios</code>
<code>src/pages/blog/index.astro</code>	→ <code>/blog</code>
<code>src/pages/blog/[slug].astro</code>	→ <code>/blog/cualquier-cosa</code>
<code>src/pages/admin/eventos.astro</code>	→ <code>/admin/eventos</code>
<code>src/pages/api/quotes.ts</code>	→ <code>/api/quotes</code> (endpoint)

src/lib/

Propósito: Configuración de librerías externas.

Archivos principales:

supabase.ts - Cliente de Supabase configurado:

```
// src/lib/supabase.ts
import { createClient } from '@supabase/supabase-js';

const supabaseUrl = import.meta.env.SUPABASE_URL;
const supabaseAnonKey = import.meta.env.SUPABASE_ANON_KEY;

export const supabase = createClient(supabaseUrl, supabaseAnonKey);
```

resend.ts - Cliente de Resend para emails:

```
// src/lib/resend.ts
import { Resend } from 'resend';

const resendApiKey = import.meta.env.RESEND_API_KEY;

export const resend = new Resend(resendApiKey);
```

src/utils/

Propósito: Funciones de utilidad reutilizables.

Archivos principales:

- `utils.ts` - Utilidades generales (cn, sleep, etc.)
- `constants.ts` - Constantes del proyecto
- `validators.ts` - Esquemas de validación Zod
- `formatters.ts` - Formateo de fechas, moneda, etc.

Ejemplo de uso:

```
import { cn } from '@utils/utils';
import { SERVICES, EVENT_TYPES } from '@utils/constants';
import { quoteSchema } from '@utils/validators';
import { formatCurrency, formatDate } from '@utils/formatters';
```

src/types/

Propósito: Tipos TypeScript compartidos.

Contenido:

```
// src/types/index.ts

// Database types (generados desde Supabase)
export interface Event {
  id: string;
  client_name: string;
  client_email: string;
  client_phone: string;
  event_type: string;
  event_date: string;
```



```

    guest_count: number;
    status: 'pending' | 'confirmed' | 'completed' | 'cancelled';
    notes?: string;
    created_at: string;
}

export interface Quote {
    id: string;
    client_name: string;
    client_email: string;
    client_phone: string;
    event_type: string;
    event_date: string;
    guest_count: number;
    message?: string;
    status: 'new' | 'contacted' | 'quoted' | 'converted' | 'declined';
    created_at: string;
}

export interface Service {
    id: string;
    name: string;
    description: string;
    price_from: number;
    features: string[];
    image_url?: string;
}

// Form types
export interface QuoteFormData {
    name: string;
    email: string;
    phone: string;
    eventType: string;
    eventDate: string;
    guestCount: number;
    message?: string;
}

export interface ContactFormData {
    name: string;
    email: string;
    phone?: string;
    subject: string;
}

```

```
message: string;
}
```

src/styles/

Propósito: Estilos CSS globales.

Archivo principal: `global.css`

- Tailwind base, components, utilities
- Estilos base HTML
- Componentes reutilizables CSS
- Animaciones custom

Ya configurado en el archivo 02B.

src/content/ (Opcional)

Propósito: Content Collections de Astro para contenido estructurado.

Configuración:

```
// src/content/config.ts
import { defineCollection, z } from 'astro:content';

const blogCollection = defineCollection({
  type: 'content',
  schema: z.object({
    title: z.string(),
    description: z.string(),
    pubDate: z.date(),
    author: z.string(),
    image: z.string().optional(),
    tags: z.array(z.string()).optional(),
  }),
});

export const collections = {
  blog: blogCollection,
};
```

Uso:

```

---
import { getCollection } from 'astro:content';

const posts = await getCollection('blog');
---

{posts.map(post => (
  <article>
    <h2>{post.data.title}</h2>
    <p>{post.data.description}</p>
  </article>
))}

```

3. SISTEMA DE RUTAS DE ASTRO

Astro usa **file-based routing**. La estructura de `src/pages/` define las rutas automáticamente.

3.1 Rutas Estáticas

```

src/pages/
├── index.astro           → /
├── servicios.astro       → /servicios
├── paquetes.astro        → /paquetes
├── portafolio.astro      → /portafolio
├── nosotros.astro        → /nosotros
├── contacto.astro        → /contacto
└── cotizacion.astro      → /cotizacion

```

3.2 Rutas con Carpetas

```

src/pages/
├── blog/
│   ├── index.astro       → /blog
│   ├── categoria1.astro  → /blog/categoria1
│   └── categoria2.astro  → /blog/categoria2
└── admin/
    ├── index.astro       → /admin

```

```
|— eventos.astro      → /admin/eventos
|— cotizaciones.astro → /admin/cotizaciones
```

3.3 Rutas Dinámicas

Parámetro único:

```
---
// src/pages/blog/[slug].astro

export async function getStaticPaths() {
  const posts = await getPosts(); // Desde Supabase o Content Collections

  return posts.map(post => ({
    params: { slug: post.slug },
    props: { post },
  }));
}

const { post } = Astro.props;
---

<h1>{post.title}</h1>
<div set:html={post.content} />
```

Múltiples parámetros:

```
---
// src/pages/eventos/[year]/[month].astro

export async function getStaticPaths() {
  return [
    { params: { year: '2025', month: '01' } },
    { params: { year: '2025', month: '02' } },
    // ...
  ];
}

const { year, month } = Astro.params;
---

<h1>Eventos de {month}/{year}</h1>
```

Rest parameters (catch-all):

```
---
// src/pages/docs/[...slug].astro
// Captura: /docs/guia/instalacion → slug = "guia/instalacion"

export async function getStaticPaths() {
  return [
    { params: { slug: 'guia/instalacion' } },
    { params: { slug: 'guia/configuracion' } },
    { params: { slug: 'api/referencia' } },
  ];
}

const { slug } = Astro.params;
const segments = slug.split('/'); // ['guia', 'instalacion']
---
```

3.4 Prioridad de Rutas

Astro resuelve rutas en este orden:

1. **Rutas estáticas** (`/about.astro`)
2. **Rutas dinámicas con parámetros** (`/[id].astro`)
3. **Rest parameters** (`/[...slug].astro`)

Ejemplo:

<code>/blog/featured.astro</code>	→ Prioridad 1 (siempre <code>/blog/featured</code>)
<code>/blog/[slug].astro</code>	→ Prioridad 2 (cualquier otro slug)
<code>/blog/[...path].astro</code>	→ Prioridad 3 (catch-all)

4. API ENDPOINTS

4.1 Crear un Endpoint

```
// src/pages/api/quotes.ts
import type { APIRoute } from 'astro';
```

```

import { supabase } from '@lib/supabase';
import { quoteSchema } from '@utils/validators';

export const POST: APIRoute = async ({ request }) => {
  try {
    // Parse request body
    const body = await request.json();

    // Validate data
    const validatedData = quoteSchema.parse(body);

    // Insert into database
    const { data, error } = await supabase
      .from('quotes')
      .insert({
        client_name: validatedData.name,
        client_email: validatedData.email,
        client_phone: validatedData.phone,
        event_type: validatedData.eventType,
        event_date: validatedData.eventDate,
        guest_count: validatedData.guestCount,
        message: validatedData.message,
        status: 'new',
      })
      .select()
      .single();

    if (error) {
      return new Response(
        JSON.stringify({ error: error.message }),
        { status: 400 }
      );
    }

    return new Response(
      JSON.stringify({ success: true, data }),
      { status: 201 }
    );
  } catch (error) {
    return new Response(
      JSON.stringify({ error: 'Invalid request data' }),
      { status: 400 }
    );
  }
}

```

```
}  
};
```

4.2 Métodos HTTP Soportados

```
// src/pages/api/events/[id].ts  
import type { APIRoute } from 'astro';  
  
// GET /api/events/123  
export const GET: APIRoute = async ({ params }) => {  
  const { id } = params;  
  // Fetch event by id  
  return new Response(JSON.stringify({ id }));  
};  
  
// PUT /api/events/123  
export const PUT: APIRoute = async ({ params, request }) => {  
  const { id } = params;  
  const body = await request.json();  
  // Update event  
  return new Response(JSON.stringify({ updated: true }));  
};  
  
// DELETE /api/events/123  
export const DELETE: APIRoute = async ({ params }) => {  
  const { id } = params;  
  // Delete event  
  return new Response(JSON.stringify({ deleted: true }));  
};  
  
// También: POST, PATCH, ALL (catch-all)
```

4.3 Ejemplo de uso desde Frontend

```
// En un componente React  
async function handleSubmit(data: QuoteFormData) {  
  const response = await fetch('/api/quotes', {  
    method: 'POST',  
    headers: {  
      'Content-Type': 'application/json',  
    },  
  },
```

```
    body: JSON.stringify(data),
  });

  if (!response.ok) {
    throw new Error('Error al enviar cotización');
  }

  const result = await response.json();
  console.log('Cotización enviada:', result.data);
}
```

CHECKLIST - Parte 1

Antes de continuar con la Parte 2 (02C-2), verifica:

Estructura creada:

- ☐ Carpetas principales creadas (`components/` , `layouts/` , `pages/` , etc.)
- ☐ Subcarpetas organizadas por tipo (layout, ui, sections, forms, admin)
- ☐ Carpeta `lib/` con archivos de configuración
- ☐ Carpeta `utils/` preparada
- ☐ Carpeta `types/` para TypeScript

Entendimiento:

- ☐ Comprendes la diferencia entre `public/` y `src/assets/`
 - ☐ Entiendes cómo funciona el file-based routing
 - ☐ Conoces cuándo usar `.astro` vs `.tsx`
 - ☐ Sabes crear rutas dinámicas
 - ☐ Comprendes cómo crear API endpoints
-

Continúa con: 02C-2 - Layouts y Componentes (Parte 2)

© 2025 La Reserva. Documentación técnica del proyecto.