



Segundo Entregable

Arquitectura y Modelamiento de Software

Juan Diego Arrieta Herrera

Valentina Ojeda Pascasio

Universidad Cooperativa de

Colombia

Ingeniería de Sistemas

Leidy Johanna González Ballesteros

Montería – Córdoba

27 de octubre de 2025

Tabla Contenido

PRESTIGE BARBERS - Segundo Entregable de Arquitectura	6
Estado Actual del Proyecto (Octubre 2025).....	6
NOTA IMPORTANTE.....	6
1. MODELO DE DOMINIO Y DATOS.....	6
<i>Entidades Implementadas Actualmente:</i>	6
<i>Entidades Planificadas (Próxima Iteración):</i>	8
<i>Relaciones Actuales:</i>	9
<i>Relaciones Futuras:</i>	9
2. DISEÑO DE API (CONTRATOS)	10
<i>Endpoints IMPLEMENTADOS y FUNCIONALES:</i>	10
<i>Endpoints PLANIFICADOS (Próxima Iteración):</i>	11
<i>Ejemplos JSON (Endpoints Actuales):</i>	13
<i>Ejemplos JSON (Endpoints Futuros):</i>	17
<i>Códigos de Error (Implementados):</i>	18
3. DECISIONES ARQUITECTÓNICAS (ADRs)	18
ADR-001: <i>Monolito Node.js con MySQL</i> (APROBADO).....	18
ADR-002: <i>Autenticación JWT con bcrypt</i> (APROBADO)	20
ADR-003: <i>Modales Personalizados vs Nativos del Navegador</i> (APROBADO).....	22
ADR-004: <i>Separación de Roles Cliente/Admin</i> (PROPUESTO).....	24
<i>Checklist N-Capas (Estado Actual):</i>	25
4. SEGURIDAD Y ROLES.....	27
<i>Modelo de Autenticación Actual:</i> (IMPLEMENTADO).....	27
<i>Política de Contraseñas:</i> (IMPLEMENTADO).....	27
<i>Tabla de Permisos por Rol (Estado Actual y Futuro):</i>	28
<i>Protección de Datos:</i> (IMPLEMENTADO)	29
<i>Manejo de Secretos:</i> (MEJORAS PENDIENTES)	30
5. IMPLEMENTACIÓN MÍNIMA (MVP)	31
<i>Funcionalidades IMPLEMENTADAS y FUNCIONANDO:</i>	31
<i>Reglas de Negocio IMPLEMENTADAS:</i>	33
<i>Estructura de Capas ACTUAL:</i>	35
└ Backend/	35
└ server.js (Controladores + Lógica + Rutas - 550 líneas).....	35

└── uploads/ (Imágenes de servicios y barberos)	35
└── package.json (Dependencias: express, mysql2, bcrypt, jwt, multer)	35
└── node_modules/	35
└── Frontend/	35
└── public/	35
└── index.html (Landing page con carousel)	35
└── js/	35
└── auth-modal.js (Sistema de autenticación - 473 líneas)	35
└── custom-modal.js (Modales personalizados - 156 líneas)	35
└── sticky-header.js (Header fijo)	35
└── carousel.js (Slider de imágenes)	35
└── I-style/	35
└── styles.css (Estilos globales + header)	35
└── auth-modal.css (Modal de autenticación)	36
└── custom-modal.css (Modales personalizados)	36
└── perfil/	36
└── index.html (Página de perfil con tabs)	36
└── perfil-styles.css (Estilos de perfil)	36
└── Cortes/	36
└── index.html (Catálogo de cortes)	36
└── C-Styles/styles.css	36
└── Barberos/	36
└── index.html (Galería de barberos)	36
└── B-Styles/styles.css	36
└── Reserva/	36
└── index.html (Página de reserva - estructura lista)	36
└── R-js/Reserva.js (Carga de servicio + barberos)	36
└── R-Styles/styles.css (Con efecto “Ver más”)	36
└── Admin/ (Sin protección auth)	36
└── index.html (Dashboard admin)	36
└── barberos.html (CRUD barberos)	37
└── gestion.html (CRUD servicios)	37
└── docs/	37
└── 01-vision-alcance.md	37

└── 02-nfrs.md	37
└── 03-c4-contexto-contenedores.md	37
└── 04-backlog.md	37
└── MODAL-AUTH-DOCUMENTATION.md	37
└── adr/.....	37
└── ADR-000-monolito-node-postgres.md	37
<i>Evidencias de Funcionamiento:</i>	38
6. PRUEBAS Y VALIDACIÓN	39
<i>Casos de Prueba EJECUTADOS:</i>	39
<i>Métricas de Rendimiento:</i>	45
<i>Próximos Riesgos Identificados:</i>	46
<i>Plan de Mitigación (Próximas Iteraciones):</i>	47
7. GUÍA DE DESPLIEGUE Y README	47
<i>Requisitos del Sistema:</i>	47
<i>Instalación Local (Paso a Paso):</i>	48
<i>Variables de Entorno (Futuro - No Implementado Aún):</i>	53
JWT.....	54
Servidor.....	54
Admin (futuro)	54
Email (futuro)	54
Scripts de Package.json:.....	54
Resumen de Rutas API (Referencia Rápida):	56
<i>Ver sección 2 para documentación completa de contratos</i>	56
- <i>POST /api/auth/register - Crear usuario</i>	56
- <i>POST /api/auth/login - Iniciar sesión</i>	56
- <i>GET /api/auth/me - Datos del usuario (requiere token)</i>	56
- <i>POST /api/auth/logout - Cerrar sesión</i>	56
- <i>POST /api/auth/change-password - Cambiar contraseña</i>	56
- <i>DELETE /api/auth/delete-account - Eliminar cuenta</i>	56
- <i>GET /api/cortes_admin - Listar cortes</i>	56
- <i>GET /api/cortes_admin/:id - Detalle de corte</i>	56
- <i>GET /api/barbas_admin - Listar barbas</i>	56
- <i>GET /api/barbas_admin/:id - Detalle de barba</i>	56
- <i>GET /api/barberos - Listar barberos</i>	56

<i>Requiere header Authorization: Bearer <token></i>	56
8. CHECKLIST DE CIERRE	58
<i>Autoevaluación contra NFRs (Requerimientos No Funcionales):</i>	59
<i>Coherencia entre Componentes:</i>	61
<i>Deuda Técnica Identificada:</i>	61
RESUMEN EJECUTIVO	62
<i>Estado del Proyecto: MVP FUNCIONAL</i>	62
<i>Funcionalidades Operativas (Octubre 2025):</i>	62
<i>Próximas Iteraciones (Planificadas):</i>	63
<i>Stack Tecnológico:</i>	63
<i>Seguridad:</i>	63
<i>Métricas de Calidad:</i>	64
<i>Roles Soportados:</i>	64
<i>Entregas:</i>	64
<i>Contexto Académico:</i>	64
FIRMA Y APROBACIÓN	65

PRESTIGE BARBERS - Segundo Entregable de Arquitectura

Estado Actual del Proyecto (Octubre 2025)

NOTA IMPORTANTE

Este documento refleja el estado ACTUAL de implementación del proyecto Prestige Barbers. Las funcionalidades aquí descritas están IMPLEMENTADAS y FUNCIONANDO. Las características marcadas como “pendientes” o “próximas iteraciones” serán desarrolladas en fases posteriores.

1. MODELO DE DOMINIO Y DATOS

Entidades Implementadas Actualmente:

USUARIOS (IMPLEMENTADA)

- id (INT, AUTO_INCREMENT, PK)
- nombre_completo (VARCHAR (255), NOT NULL)
- email (VARCHAR (255), UNIQUE, NOT NULL)
- password_hash (VARCHAR (255), NOT NULL)
- fecha_registro (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
- ultimo_acceso (TIMESTAMP, NULL)
- activo (BOOLEAN, DEFAULT true)

CORTES (IMPLEMENTADA) (Servicios de cabello)

- id (INT, AUTO_INCREMENT, PK)
- nombre (VARCHAR (255), NOT NULL)
- descripcion (TEXT)
- precio (DECIMAL (10,2))
- imagen (VARCHAR (500)) –
- tipo (VARCHAR (50)) - ‘corte’

BARBAS (IMPLEMENTADA) (Servicios de barba)

- id (INT, AUTO_INCREMENT, PK)
- nombre (VARCHAR (255), NOT NULL)
- descripcion (TEXT)
- precio (DECIMAL (10,2))
- imagen (VARCHAR (500))
- tipo (VARCHAR (50)) - ‘barba’

BARBEROS (IMPLEMENTADA)

- id (INT, AUTO_INCREMENT, PK)
- nombre (VARCHAR (255), NOT NULL)
- tipo (ENUM: ‘Peluquero’, ‘Barbero’, ‘Ambos’)
- imagen (VARCHAR (500))
- especialidades (TEXT)

Entidades Planificadas (Próxima Iteración):

RESERVAS (PENDIENTE)

- id (INT, AUTO_INCREMENT, PK)
- usuario_id (INT, FK → usuarios.id)
- barbero_id (INT, FK → barberos.id)
- servicio_id (INT)
- servicio_tipo (ENUM: ‘corte’, ‘barba’)
- fecha_hora (DATETIME, NOT NULL)
- estado (ENUM: ‘pendiente’, ‘confirmada’, ‘cancelada’, ‘completada’)
- notas (TEXT)
- created_at (TIMESTAMP)

ADMINISTRADORES (PENDIENTE)

- id (INT, AUTO_INCREMENT, PK)
- nombre (VARCHAR (255))
- email (VARCHAR (255), UNIQUE)
- password_hash (VARCHAR (255))
- rol (ENUM: 'admin', 'super_admin')
- permisos (JSON)

Relaciones Actuales:

- Usuario → Servicios (consulta, N-N implícito) (IMPLEMENTADA)
- Barbero → Servicios (filtrado por especialización) (IMPLEMENTADA)

Relaciones Futuras:

- Usuario → Reservas (1-N) (PENDIENTE)
- Barbero → Reservas (1-N) (PENDIENTE)
- Servicio (Corte/Barba) → Reservas (1-N) (PENDIENTE)
- Administrador → Gestión completa (CRUD) (PENDIENTE)

2. DISEÑO DE API (CONTRATOS)

Endpoints IMPLEMENTADOS y FUNCIONALES:

Recurso	Método	Ruta	Body/Query	Estado	Descripción
<i>Auth</i>	POST	/api/auth/register	{nombre_completo, email, password, confirm_password}	(IMPLEMENTADA)	Registrar usuario
<i>Auth</i>	POST	/api/auth/login	{email, password}	(IMPLEMENTADA)	Iniciar sesión (JWT)
<i>Auth</i>	GET	/api/auth/me	Header: Bearer	(IMPLEMENTADA)	Obtener datos del usuario
<i>Auth</i>	POST	/api/auth/logout	Header: Bearer	(IMPLEMENTADA)	Cerrar sesión
<i>Auth</i>	POST	/api/auth/change-password	{currentPassword, newPassword}	(IMPLEMENTADA)	Cambiar contraseña
<i>Auth</i>	DELETE	/api/auth/delete	{password }	(IMPLEMENTADA)	Eliminar cuenta

		e-account			
Cortes	GET	/api/cortes_admin	-	(IMPLEMENTADA)	Listar todos los cortes
Cortes	GET	/api/cortes_admin/:id	-	(IMPLEMENTADA)	Obtener corte específico
Barbas	GET	/api/barbas_admin	-	(IMPLEMENTADA)	Listar todas las barbas
Barbas	GET	/api/barbas_admin/:id	-	(IMPLEMENTADA)	Obtener barba específica
Barberos	GET	/api/barberos	-	(IMPLEMENTADA)	Listar barberos disponibles

Endpoints PLANIFICADOS (Próxima Iteración):

Recurso	Método	Ruta	Body/Query	Estado	Descripción
Admin Auth	POST	/api/admin/login	{email, password, admin_code }	(PENDIENTE)	Login de administrador

<i>Reservas</i>	POST	/api/reservas	{ usuario_id, barbero_id, servicio_id, servicio_tipo, fecha_hora, notas }	(PENDIENTE)	Crear reserva
<i>Reservas</i>	GET	/api/reservas	?usuario_id o ?barbero_id	(PENDIENTE)	Listar reservas
<i>Reservas</i>	PATCH	/api/reservas/:id/status	{ estado }	(PENDIENTE)	Actualizar estado
<i>Admin Cortes</i>	POST	/api/admin/cortes	{ nombre, descripcion, precio, imagen }	(PENDIENTE)	Crear corte (admin)
<i>Admin Cortes</i>	PUT	/api/admin/cortes/:id	{ datos }	(PENDIENTE)	Actualizar corte
<i>Admin Cortes</i>	DELETE	/api/admin/cortes/:id	-	(PENDIENTE)	Eliminar corte
<i>Admin Barberos</i>	POST	/api/admin/barberos	{ nombre, tipo, imagen }	(PENDIENTE)	Crear barbero

		beros			
<i>Admin Barberos</i>	PUT	/api/ad min/bar beros/:i d	{ datos }	(PENDIE NTE)	Actualizar barbero
<i>Admin Barberos</i>	DELETE	/api/ad min/bar beros/:i d	-	(PENDIE NTE)	Eliminar barbero

Ejemplos JSON (Endpoints Actuales):

POST /api/auth/register - REQUEST:

Json

```
{
  "nombre_completo": "Juan Diego Pérez",
  "email": "juan@example.com",
  "password": "securePass123",
  "confirm_password": "securePass123"
}
```

POST /api/auth/register - RESPONSE (200):

json

{

“success”: true,

“message”: “Usuario registrado exitosamente”,

“token”: “eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...”,

“user”: {

“id”: 1, “nombre_completo”:

“Juan Diego Pérez”,

“email”: “juan@example.com”,

“fecha_registro”: “2025-10-24T10:30:00.000Z”

}

}

POST /api/auth/login - REQUEST:

json

{

“email”: “juan@example.com”,

“password”: “securePass123”

}

POST /api/auth/login - RESPONSE (200):

json

{

“success”: true,

“message”: “Inicio de sesión exitoso”,

“token”: “eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...”,

“user”: {

“id”: 1,

“nombre_completo”: “Juan Diego Pérez”,

“email”: “juan@example.com”

}

}

GET /api/cortes_admin - RESPONSE (200):

json [

{

“id”: 1,

“nombre”: “Low Taper Fade”,

“descripcion”: “Degradado bajo clásico con transición suave”,

“precio”: 25000,

“imagen”: “/uploads/cortes/low-taper.jpg”,

“tipo”: “corte”

},

{

“id”: 2,

“nombre”: “Buzz Cut”,

“descripcion”: “Corte militar ultra corto”,

“precio”: 15000,

“imagen”: “/uploads/cortes/buzz-cut.jpg”,

“tipo”: “corte”

}

]

Ejemplos JSON (Endpoints Futuros):

POST /api/reservas - REQUEST (FUTURO):

Json

{

“usuario_id”: 1,

“barbero_id”: 3,

“servicio_id”: 1,

“servicio_tipo”: “corte”,

“fecha_hora”: “2025-11-01T15:30:00”,

“notas”: “Preferencia por degradado bajo”

}

POST /api/admin/login - REQUEST (FUTURO):

json

{

“email”: “admin@prestigebarbers.com”,

“password”: “adminSecure123”,

“admin_code”: “PB2025ADMIN”

}

Códigos de Error (Implementados):

- *400* - Datos inválidos (campos vacíos, formato incorrecto)
- *401* - No autorizado (token inválido/expirado, credenciales incorrectas)
- *404* - Recurso no encontrado (usuario/servicio no existe)
- *409* - Conflicto (email ya registrado)
- *500* - Error del servidor

3. DECISIONES ARQUITECTÓNICAS (ADRs)

ADR-001: Monolito Node.js con MySQL (APROBADO)

Fecha: 2025-10-01

Contexto:

- Proyecto universitario con equipo pequeño
- Necesidad de desarrollo rápido
- MVP funcional en corto plazo –
- Primera experiencia con arquitectura web completa

Decisión:

- Arquitectura monolítica con Node.js + Express + MySQL
- Todas las capas en un mismo proceso
- Base de datos relacional MySQL para consistencia de datos

Alternativas consideradas:

- *Microservicios*: Descartado por complejidad operacional y tiempo limitado
- *Serverless (AWS Lambda)*: Descartado por curva de aprendizaje y costos
- *MongoDB*: Descartado por necesidad de relaciones y transacciones

Consecuencias:

- (+) Desarrollo más rápido y simple
- (+) Menor complejidad operacional (un solo servidor)
- (+) Debugging más fácil
- (+) Deploy más simple
- (-) Acoplamiento de componentes
- (-) Escalabilidad limitada (vertical solamente)
- (-) Riesgo de single point of failure

Estado: Aprobado y en producción

ADR-002: Autenticación JWT con bcrypt (APROBADO)

Fecha: 2025-10-20

Contexto:

- Necesidad de autenticación segura y escalable
- Sistema sin estado (stateless) para mejor rendimiento
- Protección robusta de contraseñas en BD
- Experiencia de usuario con sesión persistente

Decisión:

- *JWT (JSON Web Tokens)* para tokens de sesión
- Expiración de *7 días* por token
- *bcrypt* con *10 rounds* para hashing de contraseñas
- Tokens almacenados en *localStorage* del cliente
- Middleware `verifyToken` para proteger rutas

Alternativas consideradas:

- *Sesiones del servidor (express-session)*: Descartado por necesidad de storage compartido y complejidad en escalamiento
- *OAuth 2.0 con proveedores externos*: Descartado por complejidad y dependencia de terceros
- *Argon2 para hashing*: Descartado por menor soporte en Node.js comparado con bcrypt

Consecuencias:

- (+) Autenticación sin estado (stateless)
- (+) Tokens seguros y verificables
- (+) Contraseñas hasheadas irreversibles
- (+) Escalabilidad horizontal sin problemas de sesión
- (+) Rendimiento óptimo (sin consultas de sesión)
- (-) Tokens no revocables hasta expiración natural
- (-) localStorage vulnerable a XSS (mitigado con CSP futuro)
- (-) Si se roba un token, es válido hasta expirar

Estado: Aprobado y funcionando

ADR-003: Modales Personalizados vs Nativos del Navegador (APROBADO)

Fecha: 2025-10-24

Contexto:

- UX inconsistente con alert(), confirm(), prompt() del navegador
- Imposibilidad de estilizar modales nativos
- Necesidad de mantener estética blanca/negro (#1b1b1b) consistente
- Mejorar profesionalismo de la interfaz de usuario
- Bloqueo de ejecución con modales nativos

Decisión:

- Desarrollar sistema de modales personalizados (clase CustomModal)
- Reemplazo total de modales nativos en *frontend de cliente* (index, perfil, cortes, barberos, reserva)
- *Mantener modales nativos en panel de administración* para simplicidad
- Animaciones suaves con cubic-bezier(0.25, 0.46, 0.45, 0.94)
- Soporte de teclado (ESC, Enter)

Alternativas consideradas:

- *Mantener modales nativos*: Descartado por UX pobre y falta de control visual
- *Librería externa (SweetAlert2, Toastrify)*: Descartado por dependencia extra y personalización limitada
- *Framework completo (Bootstrap Modal)*: Descartado por peso y sobrecarga de features

Consecuencias:

- (+) Control total del diseño y animaciones
- (+) Experiencia de usuario consistente y profesional
- (+) Animaciones fluidas y elegantes (estilo Louis Vuitton)
- (+) Soporte completo de teclado
- (+) Async/await compatible (no bloquea ejecución)
- (-) ~6KB adicionales de código (custom-modal.js + custom-modal.css)
- (-) Mantenimiento propio del código
- (-) Testing adicional requerido

Estado: Aprobado e implementado

ADR-004: Separación de Roles Cliente/Admin (PROUESTO)

Fecha: 2025-10-24

Contexto:

- Necesidad de panel administrativo para gestión de barbería
- Diferentes niveles de acceso y permisos
- Actualmente solo existe rol de Cliente
- Panel de admin ya existe en /Admin/ pero sin autenticación

Decisión (Propuesta):

- Crear tabla administradores separada de usuarios
- Implementar endpoint /api/admin/login con código adicional de seguridad
- Middleware verificarAdmin para proteger rutas administrativas
- Redireccionamiento a index.html tras login exitoso
- Mantener sesiones separadas (admin no puede acceder a perfil de cliente y viceversa)

Alternativas consideradas:

- *Campo “rol” en tabla usuarios*: Descartado por mezclar conceptos (clientes y empleados)
- *Sistema de permisos granular (RBAC)*: Pospuesto para fase posterior

Consecuencias esperadas:

- (+) Separación clara de responsabilidades
- (+) Mayor seguridad (código admin adicional)
- (+) Panel de administración protegido
- (-) Dos flujos de autenticación separados
- (-) Duplicación parcial de lógica de auth

Estado: Propuesto - Implementación en próxima iteración

Checklist N-Capas (*Estado Actual*):

- Controladores sin lógica de negocio (solo validación y respuesta HTTP)
- Uso de bcrypt para seguridad (no SQL directo de passwords)
- *Repositorios mezclados con controladores* (todo en server.js - refactorizar pendiente)
- *Sin capa de servicios explícita* (lógica de negocio en controladores)

Plan de Refactorización (Futuro):

Backend/

|

 └── controllers/

 └── *Manejo de solicitudes HTTP, validación de datos y construcción de respuestas.*

|

 └── services/

 └── *Lógica de negocio y reglas del dominio. Actúa como intermediario entre controladores y repositorios.*

|

 └── repositories/

 └── *Acceso a datos y consultas a la base de datos (SQL, ORM, etc.).*

|

 └── middlewares/

 └── *Funciones intermedias para autenticación, validación global y manejo de errores.*

*

|

 └── routes/

└— *Definición de rutas y endpoints de la API. Conecta las rutas con los controladores correspondientes. *

4. SEGURIDAD Y ROLES

Modelo de Autenticación Actual: (IMPLEMENTADO)

Tipo: JWT (JSON Web Tokens)

- *Algoritmo:* HS256
- *Secret Key:* prestige_barbers_secret_key_2025 ▲ (debe moverse a .env)
- *Expiración:* 7 días (604800 segundos)
- *-Storage Cliente:* localStorage
- *Header:* Authorization: Bearer <token>

Política de Contraseñas: (IMPLEMENTADO)

- Mínimo 6 caracteres (validación frontend + backend)
- Hashing con bcrypt (10 rounds = 2^{10} iteraciones)
- Confirmación requerida en registro
- Validación de contraseña actual para cambios y eliminación de cuenta
- Nunca se retornan contraseñas en respuestas API

Tabla de Permisos por Rol (Estado Actual y Futuro):

Rol	Estado	Permisos Actuales	Permisos Futuros
<i>Client e (Usuar io)</i>	Implementado	- Ver catálogo de servicios- Ver barberos- Gestionar perfil (editar, cambiar contraseña, eliminar cuenta)- Cerrar sesión	- Crear reservas- Ver historial de reservas- Cancelar reservas- Dejar reseñas
<i>Barber o</i>	Planificado	- Ninguno (rol no implementado)	- Ver agenda personal- Confirmar/rechazar citas- Ver detalles de clientes asignados- Marcar servicios como completados
<i>Admin istrado r</i>	Planificado	- Acceso a /Admin/ sin auth (temporal)	- CRUD completo de barberos- CRUD completo de servicios (cortes/barbas)- Ver todas las reservas- Gestionar usuarios- Ver reportes y estadísticas- Configurar disponibilidad
<i>Super Admin</i>	Planteado	- No existe	- Gestionar otros administradores- Configuración del sistema- Backup de datos

Protección de Datos: (IMPLEMENTADO)

Contraseñas:

- Nunca almacenadas en texto plano
- Hash con bcrypt antes de INSERT/UPDATE
- Verificación con bcrypt.compare() (no comparación directa)

Tokens JWT:

- Verificado en rutas protegidas (middleware verificar Token)
- Expira automáticamente después de 7 días
- No hay lista negra (blacklist) para revocación temprana

Operaciones Críticas:

- Eliminación de cuenta requiere *contraseña + doble confirmación*
- Cambio de contraseña requiere *contraseña actual*
- No se expone información sensible en errores

Pendientes de Implementar:

- Rate limiting (prevenir brute force)
- Verificación de email
- Recuperación de contraseña por email
- HTTPS en producción (actualmente HTTP local)

- Política de Content Security Policy (CSP)

Manejo de Secretos: (MEJORAS PENDIENTES)

Estado Actual:

javascript // Backend/server.js (línea 15)

```
const JWT_SECRET = 'prestige_barbers_secret_key_2025'; // ⚡ Hardcoded
```

// DB Connection (línea 10)

```
const db = mysql.createPool({
```

```
    host: 'localhost',
```

```
    user: 'root',
```

```
    password: '', // Sin contraseña en desarrollo
```

```
    database: 'Barberia'
```

```
});
```

Configuración Futura (.env):

env

Próxima implementación

DB_HOST=localhost

DB_USER=root

DB_PASSWORD=secure_password_here

DB_NAME=Barberia

JWT_SECRET=random_secure_key_generated

JWT_EXPIRES_IN=7d

PORt=3000

NODE_ENV=development

ADMIN_SECRET_CODE=admin_verification_code

5. IMPLEMENTACIÓN MÍNIMA (MVP)

Funcionalidades IMPLEMENTADAS y FUNCIONANDO:

1. *Sistema de Autenticación Completo:*

- Registro de usuarios con validaciones (email único, passwords coinciden, longitud mínima)
- Login con generación de JWT
- Persistencia de sesión (localStorage + token verificación)
- Logout con limpieza de sesión
- Cambio de contraseña (requiere contraseña actual)
- Eliminación de cuenta (doble confirmación + password)
- Protección de rutas con middleware verificarToken

2. Gestión de Perfil de Usuario:

- Visualización de datos personales (nombre, email, fecha de registro)
- Edición de información (actualización en localStorage)
- Avatar con inicial del nombre
- Badge de usuario logueado en header
- Tabs de navegación (Mi Información, Mis Reservas, Configuración)

3. Catálogo de Servicios:

- Visualización de cortes disponibles con imágenes y descripciones
- Visualización de servicios de barba
- Efecto “Ver más” elegante (estilo Louis Vuitton) en descripciones largas
- Navegación por categorías
- Imágenes relacionadas y galería

4. Sistema de Barberos:

- Listado de barberos disponibles
- Filtrado por especialización (Peluquero, Barbero, Ambos)
- Selección de barbero en página de reserva
- Imágenes y perfiles de barberos

5. Interfaz de Usuario:

- Modal de autenticación universal (funciona en todas las páginas)
- Modales personalizados para confirmaciones (alert, confirm, prompt)
- Diseño responsive blanco/negro (#1b1b1b)
- Animaciones fluidas con cubic-bezier
- Header sticky con navegación
- Carousel de imágenes en home

6. Panel de Administración (Sin Auth - Temporal):

- Interfaz de gestión de barberos en barberos.html
- Interfaz de gestión de servicios en gestion.html
- Acceso sin autenticación (pendiente proteger)

Reglas de Negocio IMPLEMENTADAS:

RN-001: Prevención de Usuarios Duplicados

- Validación de email único en registro
- Query SQL: SELECT * FROM usuarios WHERE email = ?
- Respuesta HTTP 409 Conflict si email ya existe
- Mensaje: “El correo ya está registrado”

RN-002: Protección de Contraseñas

- Hash con bcrypt (10 rounds) antes de almacenar
- Nunca se retorna password_hash en respuestas
- Verificación con bcrypt.compare() en login

RN-003: Validación de Token Vigente

- Middleware verifica firma JWT y expiración
- Respuesta 401 si token inválido o expirado
- Decodifica payload para obtener usuario_id

RN-004: Filtrado de Barberos por Especialización

- En página de reserva, filtra según tipo de servicio
- Corte → muestra Peluqueros y Ambos
- Barba → muestra Barberos y Ambos

Estructura de Capas ACTUAL:

Prestige-Barbers-main/

```
| └── Backend/  
|   | └── server.js (Controladores + Lógica + Rutas - 550 líneas)  
|   | └── uploads/ (Imágenes de servicios y barberos)  
|   | └── package.json (Dependencias: express, mysql2, bcrypt, jwt, multer)  
|   └── node_modules/  
└── Frontend/  
    | └── public/  
    |   | └── index.html (Landing page con carousel)  
    |   | └── js/  
    |   |   | └── auth-modal.js (Sistema de autenticación - 473 líneas)  
    |   |   | └── custom-modal.js (Modales personalizados - 156 líneas)  
    |   |   | └── sticky-header.js (Header fijo)  
    |   |   | └── carousel.js (Slider de imágenes)  
    |   └── I-style/  
    |   | └── styles.css (Estilos globales + header)
```

```
|   |   └── auth-modal.css (Modal de autenticación)
|   |   └── custom-modal.css (Modales personalizados)
|   └── perfil/
|       |   ├── index.html (Página de perfil con tabs)
|       |   └── perfil-styles.css (Estilos de perfil)
|   └── Cortes/
|       |   ├── index.html (Catálogo de cortes)
|       |   └── C-Styles/styles.css
|   └── Barberos/
|       |   ├── index.html (Galería de barberos)
|       |   └── B-Styles/styles.css
|   └── Reserva/
|       |   ├── index.html (Página de reserva - estructura lista)
|       |   └── R-js/Reserva.js (Carga de servicio + barberos)
|       |   └── R-Styles/styles.css (Con efecto “Ver más”)
|   └── Admin/ (Sin protección auth)
|       └── index.html (Dashboard admin)
```

```
|      └── barberos.html (CRUD barberos)
|
|      └── gestion.html (CRUD servicios)
|
└── docs/
    |
    |   ├── 01-vision-alcance.md
    |
    |   ├── 02-nfrs.md
    |
    |   ├── 03-c4-contexto-contenedores.md
    |
    |   ├── 04-backlog.md
    |
    |   └── MODAL-AUTH-DOCUMENTATION.md
    |
    └── adr/
        └── ADR-000-monolito-node-postgres.md
```

Evidencias de Funcionamiento:

Captura 1: Registro de Usuario

- Input: nombre=“Juan Pérez”, email=“juan@test.com”, password=“123456”
- Output: JWT generado, usuario en BD, sesión iniciada
- Status: Funcionando

Captura 2: Login con Badge

- Usuario logueado ve su inicial en círculo negro en header
- Click en badge redirige a /perfil/
- Status: Funcionando

Captura 3: Perfil con Tabs

- Tab “Mi Información” muestra datos editables
- Tab “Configuración” permite cambio de contraseña y eliminación de cuenta
- Status: Funcionando

Captura 4: Modales Personalizados

- Confirmación de eliminación de cuenta con 2 modales + prompt de password
- Diseño blanco/negro consistente
- Status: Funcionando

Captura 5: Catálogo de Servicios

- Grid de cortes con imágenes
- Click en servicio carga /Reserva/?id=X&type=corte
- Status: Funcionando

Captura 6: Efecto “Ver más”

- Descripción limitada a 3 líneas con gradient fade
- Botón “Ver más” expande suavemente
- Status: Funcionando

6. PRUEBAS Y VALIDACIÓN

Casos de Prueba EJECUTADOS:

CASO 1: Registro de Usuario (Happy Path) COMPLETADO

- *Precondición:* Email no existe en BD
- *Input:*

json

{

“nombre_completo”: “Juan Pérez”,

“email”: “juan@test.com”,

“password”: “123456”,

```
        "confirm_password": "123456"
```

```
}
```

- *Esperado:*
 - Status 200
 - Usuario insertado en BD con password hasheado
 - JWT retorna válido por 7 días
 - localStorage con auth_token y user_data
 - Redirección automática o cierre de modal
- *Resultado:* COMPLETADO - Usuario registrado exitosamente
- *Evidencia:* Query BD muestra registro con bcrypt hash de 60 caracteres

CASO 2: Login con Credenciales Incorrectas (Error Path) COMPLETADO

- *Precondición:* Usuario existe en BD

- *Input:*

```
json
```

```
{
```

```
    "email": "juan@test.com",
```

```
    "password": "wrong_password"
```

```
}
```

- *Esperado:*
 - Status 401 Unauthorized
 - Mensaje: “Contraseña incorrecta”
 - No se genera token
 - Modal personalizado muestra error
- *Resultado:* COMPLETADO - Error manejado correctamente
- *Evidencia:* bcrypt.compare() retorna false, respuesta 401 enviada

CASO 3: Registro con Email Duplicado (Error Path) COMPLETADO

- *Precondición:* Email “juan@test.com” ya existe
- *Input:*

json

{

“nombre_completo”: “Otro Usuario”,

“email”: “juan@test.com”,

“password”: “newpass”,

“confirm_password”: “newpass”

}

- *Esperado:*
 - Status 409 Conflict
 - Mensaje: “El correo ya está registrado”
 - No se crea usuario duplicado
- *Resultado:* COMPLETADO - Conflicto detectado
- *Evidencia:* Query SELECT encuentra email existente, INSERT no ejecutado

CASO 4: Cambio de Contraseña (Happy Path) COMPLETADO

- *Precondición:* Usuario logueado con token válido

- *Input:*

json

{

“currentPassword”: “123456”,

“newPassword”: “newSecure789”

}

- *Esperado:*

- Status 200
- Contraseña actualizada en BD con nuevo hash bcrypt
- Mensaje: “Contraseña actualizada exitosamente”
- Campos de input limpiados

- *Resultado:* COMPLETADO - Hash actualizado correctamente
- *Evidencia:* BD muestra nuevo password_hash diferente al anterior

CASO 5: Eliminación de Cuenta (Happy Path) COMPLETADO

- *Precondición:* Usuario logueado
- *Input:*
 - Primera confirmación: Aceptar
 - Segunda confirmación: Aceptar
 - Prompt password: “123456” (correcta)

Esperado:

- Status 200
- Usuario eliminado de BD (DELETE FROM usuarios WHERE id = ?)
- localStorage limpiado
- Redirección a home después de 1 segundo
- *Resultado:* COMPLETADO - Cuenta eliminada permanentemente
- *Evidencia:* Query BD no encuentra usuario, localStorage vacío

CASO 6: Acceso a Ruta Protegida sin Token (Error Path) COMPLETADO

- *Precondición:* No hay token en localStorage
- *Request:* GET /api/auth/me sin header Authorization
- *Esperado:*
 - Status 401
 - Mensaje: “Token no proporcionado”
 - Redirección a página de login

Resultado: COMPLETADO - Acceso denegado

Evidencia: Middleware verificarToken retorna error antes de llegar al controlador

CASO 7: Filtrado de Barberos por Tipo de Servicio COMPLETADO

- *Precondición:* BD tiene 5 barberos (2 Peluqueros, 2 Barberos, 1 Ambos)
- *Input:* Acceso a /Reserva/?id=1&type=corte
- *Esperado:*
 - Solo se muestran Peluqueros y Ambos (3 barberos)
 - Barberos de tipo “Barbero” no aparecen
- *Resultado:* COMPLETADO - Filtrado correcto
- *Evidencia:* Renderizado de solo 3 botones de barbero

CASO 8: Efecto “Ver más” con Texto Corto COMPLETADO

- *Precondición:* Servicio con descripción de 2 líneas
- *Esperado:*
 - No se muestra botón “Ver más”
 - No se muestra gradient fade
 - Texto completamente visible
- *Resultado:* COMPLETADO - Comportamiento inteligente
- *Evidencia:* Botón tiene clase .hidden, fade también oculto

Métricas de Rendimiento:

Métrica	Objetivo	Resultado Actual	Estado
Tiempo de respuesta login (p95)	< 500ms	~280ms	Óptimo
Tiempo de respuesta registro (p95)	< 800ms	~450ms (bcrypt)	Óptimo
Tiempo de	< 1s	~300ms	Óptimo

carga catálogo			
Disponibilidad (local)	99.9%	100%	Óptimo
Tasa de errores	< 1%	~0.2% (solo errores de validación esperados)	Óptimo
Tamaño de modales personalizados	< 10KB	~6KB (custom-modal.js + .css)	Óptimo

Próximos Riesgos Identificados:

CRÍTICO:

- *Sin rate limiting:* Vulnerable a ataques de fuerza bruta en login
- *Secret key hardcodeado:* Compromete seguridad si se expone código

MEDIO:

- *Tokens no revocables:* Si se roba un token, es válido hasta expirar (7 días)
- *localStorage vulnerable a XSS:* Si hay inyección de scripts, token puede ser robado
- *Sin verificación de email:* Usuarios falsos pueden registrarse

BAJO:

- *Panel admin sin auth:* Cualquiera puede acceder a /Admin/ (temporal, en desarrollo)

- *Sin recuperación de contraseña:* Usuario bloqueado si olvida password
- *Sin logging de accesos:* Difícil auditar actividad sospechosa

Plan de Mitigación (Próximas Iteraciones):

1. *Implementar rate limiting* con express-rate-limit (5 intentos por 15 min)
2. *Mover secretos a .env* con dotenv
3. *Implementar refresh tokens* (token de acceso corto + refresh de 30 días)
4. *Agregar CSP headers* para prevenir XSS
5. *Proteger panel admin* con autenticación separada
6. *Implementar sistema de recuperación de contraseña* por email

7. GUÍA DE DESPLIEGUE Y README

Requisitos del Sistema:

- *Node.js:* v16.0.0 o superior (recomendado v18 LTS)
- *MySQL:* 8.0 o superior
- *npm:* 7.0.0 o superior (incluido con Node.js)
- *Navegador:* Chrome 90+, Firefox 88+, Edge 90+ (para módulos ES6)
- *Sistema Operativo:* Windows 10/11, macOS 11+, Linux (Ubuntu 20.04+)

Instalación Local (Paso a Paso):

1. Clonar Repositorio:

- bash
- git clone https://github.com/dr3amy-diamonds/Prestige-Barbers.git
- cd Prestige-Barbers-main

2. Configurar Base de Datos MySQL:

- sql
 - Conectar a MySQL
 - mysql -u root -p
- Crear base de datos

```
CREATE DATABASE Barberia CHARACTER SET utf8mb4 COLLATE  
utf8mb4_unicode_ci;
```

-- Usar la base de datos

```
USE Barberia;
```

-- Crear tabla usuarios

```
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre_completo VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ultimo_acceso TIMESTAMP NULL,
    activo BOOLEAN DEFAULT true
);
```

-- Crear tabla barberos (ejemplo)

```
CREATE TABLE barberos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    tipo ENUM('Peluquero', 'Barbero', 'Ambos') NOT NULL,
    imagen VARCHAR(500),
    especialidades TEXT
);
```

```
-- Crear tabla cortes (ejemplo)
```

```
CREATE TABLE cortes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    descripcion TEXT,
    precio DECIMAL(10,2),
    imagen VARCHAR(500),
    tipo VARCHAR(50) DEFAULT 'corte'
);
```

```
-- Insertar datos de ejemplo (opcional)
```

```
INSERT INTO barberos (nombre, tipo, imagen) VALUES
('Carlos Méndez', 'Ambos', '/uploads/barberos/carlos.jpg'),
('Ana Rodríguez', 'Peluquero', '/uploads/barberos/ana.jpg');
```

```
```3. Instalar Dependencias del Backend:
```

- bash

- cd Backend
- npm install

Esto instalará:

- express@5.1.0 (servidor web)
- mysql2@3.15.1 (driver MySQL)
- bcrypt@5.1.1 (hashing de contraseñas)
- jsonwebtoken@9.0.2 (JWT)
- multer@2.0.2 (manejo de archivos)
- cors@2.8.5 (CORS habilitado)

#### 4. Verificar Configuración de BD:

```
const db = mysql.createPool({

 host: 'localhost', // ← Cambiar si BD en otro host

 user: 'root', // ← Usuario MySQL

 password: "", // ← Contraseña MySQL (vacía en desarrollo)

 database: 'Barberia', // ← Nombre de la BD

 waitForConnections: true,

 connectionLimit: 10,

 queueLimit: 0
```

```
});
```

*5. Iniciar Servidor Backend:*

bash node

server.js

Salida esperada:

Servidor corriendo en http://localhost:3000

Conexión exitosa a la base de datos MySQL

*6. Abrir Frontend:*

- *Opción A (con extensión de VS Code):*
  - Instalar “Live Server” en VS Code
  - Abrir index.html
  - Click derecho → “Open with Live Server”
  - Abre en http://127.0.0.1:5500/Frontend/public/index.html
- *Opción B (directamente en navegador):*
  - Abrir index.html en Chrome/Firefox
  - Algunas funcionalidades pueden fallar por CORS (usar opción A)

*7. Verificar Funcionamiento:*

- [ ] Página principal carga con header y carousel
- [ ] Click en icono de usuario abre modal de login/registro
- [ ] Registro crea usuario en BD
- [ ] Login genera token y muestra badge con inicial
- [ ] Click en badge redirige a /perfil/
- [ ] Servicios cargan desde BD

*Variables de Entorno (Futuro - No Implementado Aún):*

Crear archivo .env en Backend:

env

# Base de datos

DB\_HOST=localhost

DB\_USER=root D

B\_PASSWORD=tu\_password\_segura

DB\_NAME=Barberia

DB\_PORT=3306

## **JWT**

JWT\_SECRET=tu\_clave\_secreta\_random\_aqui\_min\_32\_caracteres

JWT\_EXPIRES\_IN=7d

## **Servidor**

PORt=3000

NODE\_ENV=development

## **Admin (futuro)**

ADMIN\_SECRET\_CODE=codigo\_verificacion\_admin\_2025

## **Email (futuro)**

SMTP\_HOST=smtp.gmail.com

SMTP\_PORT=587

SMTP\_USER=noreply@prestigebarbers.com

SMTP\_PASSWORD=app\_specific\_password

## **Scripts de Package.json:**

json

{

  "name": "prestige-barbers-backend",

  "version": "1.0.0",

  "scripts": {

```
"start": "node server.js",

"dev": "nodemon server.js",

"test": "echo \"No tests yet\" && exit 0"

},

"dependencies": {

 "express": "^5.1.0",

 "mysql2": "^3.15.1",

 "bcrypt": "^5.1.1",

 "jsonwebtoken": "^9.0.2",

 "multer": "^2.0.2",

 "cors": "^2.8.5"
}

},

"devDependencies": {

 "nodemon": "^3.0.1"
}
```

## **Resumen de Rutas API (Referencia Rápida):**

*Ver sección 2 para documentación completa de contratos*

### *Autenticación:*

- *POST /api/auth/register - Crear usuario*
- *POST /api/auth/login - Iniciar sesión*
- *GET /api/auth/me - Datos del usuario (requiere token)*
- *POST /api/auth/logout - Cerrar sesión*
- *POST /api/auth/change-password - Cambiar contraseña*
- *DELETE /api/auth/delete-account - Eliminar cuenta*

### *Servicios (público):*

- *GET /api/cortes\_admin - Listar cortes*
- *GET /api/cortes\_admin/:id - Detalle de corte*
- *GET /api/barbas\_admin - Listar barbas*
- *GET /api/barbas\_admin/:id - Detalle de barba*
- *GET /api/barberos - Listar barberos*

*Requiere header Authorization: Bearer <token>*

### *Troubleshooting Común:*

Error: “Cannot GET /”

- *Causa:* Servidor no iniciado
- *Solución:* Ejecutar node server.js en /Backend

Error: “ER\_ACCESS\_DENIED\_ERROR”

- *Causa:* Credenciales de MySQL incorrectas
- *Solución:* Verificar usuario/password en server.js línea 10-15

Error: “ER\_BAD\_DB\_ERROR”

- *Causa:* Base de datos “Barberia” no existe
- *Solución:* Ejecutar CREATE DATABASE Barberia; en MySQL

Error: “CORS policy blocked”

- *Causa:* Frontend y backend en dominios diferentes

*Solución:* Ya implementado CORS en servidor, verificar que backend esté corriendo

### *Modal de login no abre:*

- *Causa:* Script auth-modal.js no cargado
- *Solución:* Verificar que esté en HTML

### *Token expirado:*

- *Causa:* 7 días pasados desde login
- *Solución:* Logout y login nuevamente

## 8. CHECKLIST DE CIERRE

Ítem	Estado	Evidencia	Notas
<i>Diagrama y diccionario de datos completos</i>	COMPLETADO	Sección 1	4 entidades implementadas, 4 planificadas
<i>Contrato de API documentado</i>	COMPLETADO	Sección 2	11 endpoints funcionando, 10 planificados
<i>2–3 ADR nuevos creados</i>	COMPLETADO	Sección 3	3 ADRs aprobados, 1 propuesto
<i>Roles y seguridad definidos</i>	PENDIENTE	Sección 4	Solo rol Cliente implementado, Admin y Barbero planificados
<i>MVP funcional con 1 regla de negocio</i>	COMPLETADO	Sección 5	4 reglas implementadas (duplicados, hash, token, filtrado)
<i>Pruebas y evidencias</i>	COMPLETADO	Sección 6	8 casos de prueba ejecutados,

<i>incluidas</i>			métricas de rendimiento
<i>README y guías de ejecución</i>	COMPLETADO	Sección 7	Instalación paso a paso, troubleshooting
<i>Código versionado en GitHub</i>	COMPLETADO	-	Repositorio dr3amy-diamonds/Prestig e-Barbers
<i>Documentación en /docs</i>	COMPLETADO	-	5 archivos MD en carpeta docs/
<i>Estructura de carpetas consistente</i>	COMPLETADO	-	Backend/, Frontend/, docs/, adr/

#### *Autoevaluación contra NFRs (Requerimientos No Funcionales):*

Atributo	Objetivo	Estado Actual	Métrica	Cumplimiento
<i>Seguridad</i>	Datos protegidos, autenticación robusta	Implementado	JWT + bcrypt, contraseñas hasheadas	90%
<i>Usabilidad</i>	Interfaz intuitiva, <=	Implementado	Modales elegantes,	95%

	3 clicks para acción		navegación simple	
<i>Rendimiento</i>	Respuesta API <= 500ms (p95)	Óptimo	280ms login, 450ms registro	100%
<i>Mantenibilidad</i>	Código claro, documentado	Mejorable	Todo en server.js, sin capas separadas	60%
<i>Escalabilidad</i>	Soporte de 100+ usuarios concurrentes	No probado	Monolito sin load balancing	50%
<i>Disponibilidad</i>	Uptime >= 99%	En local	100% en desarrollo	N/A
<i>Compatibilidad</i>	Funciona en Chrome, Firefox, Edge	Probado	Compatible con ES6	100%

### *Coherencia entre Componentes:*

#### *Modelo → API → Código:*

- Tabla usuarios tiene campos que se mapean exactamente a endpoints /api/auth/\*
- Responses JSON coinciden con estructura de BD
- Frontend consume API según contrato documentado

#### *Documentación → Implementación:*

- ADRs reflejan decisiones tomadas en código
- Sección 5 coincide con estructura de carpetas real
- Ejemplos JSON son respuestas reales de API

#### *Versionado y Nomenclatura:*

- Archivos nombrados consistentemente (kebab-case para docs, camelCase para JS)
- Commits descriptivos en GitHub
- Fecha de documentación: 24-Oct-2025

### *Deuda Técnica Identificada:*

#### Alta Prioridad:

1. *Refactorizar server.js* → Separar en capas (controllers, services, repositories)
2. *Mover secretos a .env* → Implementar dotenv
3. *Proteger panel admin* → Autenticación separada

### Media Prioridad:

4. *Implementar rate limiting* → Prevenir brute force
5. *Agregar refresh tokens* → Mejorar seguridad
6. *Sistema de logging* → Winston o Morgan

### Baja Prioridad:

7. *Testing automatizado* → Jest + Supertest
8. *CI/CD pipeline* → GitHub Actions
9. *Dockerización* → Dockerfile + docker-compose

## RESUMEN EJECUTIVO

### ***Estado del Proyecto: MVP FUNCIONAL***

*Prestige Barbers* es una aplicación web moderna para gestión de reservas en barbería, actualmente en fase de *MVP completado* con las siguientes características implementadas:

### ***Funcionalidades Operativas (Octubre 2025):***

- Sistema de autenticación completo (registro, login, logout, cambio de contraseña, eliminación de cuenta)
- Gestión de perfil de usuario con interfaz de tabs
- Catálogo interactivo de servicios (cortes y barbas) con efecto “Ver más” elegante
- Sistema de visualización de barberos con filtrado por especialización
- UI/UX premium con modales personalizados estilo Louis Vuitton

- Persistencia de sesión con JWT (7 días)
- Diseño responsive blanco/negro (#1b1b1b)

### ***Próximas Iteraciones (Planificadas):***

- Sistema de reservas con calendario (entidad RESERVAS)
- Autenticación de administrador con acceso a panel protegido
- CRUD completo de servicios y barberos desde admin
- Dashboard de barbero con agenda personal
- Sistema de notificaciones por email
- Recuperación de contraseña
- Reseñas y valoraciones de clientes

### ***Stack Tecnológico:***

- *Backend:* Node.js 18 + Express 5.1.0 + MySQL 8.0
- *Frontend:* Vanilla JavaScript ES6 + HTML5 + CSS3 (Montserrat font)
- *Seguridad:* JWT (HS256) + bcrypt (10 rounds)
- *Arquitectura:* Monolito en 3 capas (pendiente refactorización)

### ***Seguridad:***

- Contraseñas hasheadas con bcrypt (irreversible)
- Tokens JWT con expiración de 7 días
- Validación de contraseña para operaciones críticas
- Protección contra usuarios duplicados (email único)
- Pendiente: Rate limiting, HTTPS, CSP headers, blacklist de tokens

### *Métricas de Calidad:*

- *Rendimiento:* p95 < 500ms ( 280ms promedio)
- *Disponibilidad:* 100% en entorno local
- *Cobertura de pruebas:* 8 casos manuales ejecutados ( 100% COMPLETADO)
- *Código:* ~1,200 líneas JavaScript, ~800 líneas CSS
- *Documentación:* 5 archivos MD, 3 ADRs

### *Roles Soportados:*

- *Cliente (Usuario):* Registro, login, perfil, visualización de servicios
- *Barbero:* Agenda, confirmación de citas (pendiente)
- *Administrador:* CRUD servicios, barberos, usuarios (pendiente auth)

### *Entregas:*

- *Semana 15:* Sustentación técnica (20 min) - Demo en vivo + presentación ADRs
- *Semana 16:* Entrega final PDF + código completo

### *Contexto Académico:*

- *Universidad:* Cooperativa de Colombia
- *Programa:* Ingeniería de Sistemas
- *Curso:* Arquitectura de Software
- *Entregable:* Segundo entregable (Arquitectura implementable)
- *Fecha:* Octubre 2025

## FIRMA Y APROBACIÓN

*Documento preparado por:* Equipo Prestige Barbers

*Fecha de elaboración:* 24 de octubre de 2025

*Versión:* 1.0 (Estado actual al 24-Oct-2025)

*Próxima revisión:* Noviembre 2025 (tras implementar sistema de reservas)

### Estado de aprobación:

- Arquitectura revisada y aprobada
- Código funcional validado
- Documentación técnica completa
- Panel de administrador con autenticación (pendiente)
- Sistema de reservas (pendiente)
- Testing automatizado (pendiente)