Travel Planner Frontend Design Document

> ***Version:*** *1.0.0*
> ***Framework:*** *Next.js 16 (App Router)*
> ***Last Updated:*** *January 2025*

Table of Contents

---

1. Overview

The Travel Planner Frontend is a Next.js application providing user interfaces for:

- **Public Users**: Browse travel packages, hotels, transport, and food services
- **Customers**: Register, login, create bookings (catalog or custom), manage profile, process payments
- **Administrators**: Dashboard analytics, manage all bookings/payments/customers, CRUD operations for catalogs and services

Key Features

Feature	Description
Package Browsing	View curated travel packages with pricing
Custom Bookings	Build custom trips with individual services
Catalog Bookings	Book pre-made travel packages
Payment Processing	Two-step payment flow (process → complete)
Admin Dashboard	Business analytics and management
Responsive Design	Mobile-first responsive layouts

---

2. Architecture

Application Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                         Browser (Client)                        │
└─────────────────────────────────────────────────────────────────┘
                                │
                                ▼
┌─────────────────────────────────────────────────────────────────┐
│                      Next.js App Router                         │
│   ┌─────────────┐  ┌─────────────┐  ┌─────────────────────┐    │
│   │   Pages     │  │ Components  │  │      Layouts        │    │
│   │  (app/*)    │  │  (ui/*)     │  │  (layout.tsx)       │    │
│   └─────────────┘  └─────────────┘  └─────────────────────┘    │
│                                                                 │
│   ┌─────────────┐  ┌─────────────┐  ┌─────────────────────┐    │
│   │   lib/      │  │  Context    │  │   Route Guards      │    │
│   │  (api.ts)   │  │  (auth)     │  │  (protected-route)  │    │
│   └─────────────┘  └─────────────┘  └─────────────────────┘    │
└─────────────────────────────────────────────────────────────────┘
                                │
                                ▼ (Axios HTTP)
┌─────────────────────────────────────────────────────────────────┐
│                  Backend API (localhost:3001)                   │
│                     Express.js + Oracle DB                      │
└─────────────────────────────────────────────────────────────────┘
```

Data Flow

```
User Action → Page Component → API Helper (lib/api.ts) → Backend API
                                                              │
UI Update ← State Update ← Response ←──────────────────────────┘
```

---

3. Tech Stack

Core Dependencies

Package	Version	Purpose
next	16.x	React framework with App Router
react	19.x	UI library
typescript	5.x	Type safety
tailwindcss	4.x	Utility-first CSS
axios	1.x	HTTP client

UI Components (shadcn/ui)

Component	Usage
button	Actions, form submissions, navigation

card	Content containers, stats display
input	Form text inputs
label	Form field labels
select	Dropdown selections
table	Data listings (bookings, payments, etc.)
badge	Status indicators, tags
dialog	Modal dialogs for confirmations
alert-dialog	Destructive action confirmations
tabs	Content organization
separator	Visual dividers
skeleton	Loading placeholders
calendar	Date pickers
dropdown-menu	User menu, actions menu
avatar	User profile display
form	Form validation wrapper
textarea	Multi-line text input
sonner	Toast notifications

Utility Libraries

Package	Purpose
jwt-decode	Decode JWT tokens client-side
date-fns	Date formatting and manipulation
lucide-react	Icon library
zod	Schema validation
react-hook-form	Form state management

---

4. Project Structure

```
frontend/
├── public/                     # Static assets
├── src/
│   ├── app/                    # App Router pages
│   │   ├── layout.tsx          # Root layout (AuthProvider, Navbar, Toaster)
│   │   ├── page.tsx            # Home page (catalog listing)
│   │   ├── globals.css         # Global styles
│   │   │
│   │   ├── login/              # Customer login
│   │   │   └── page.tsx
│   │   ├── register/           # Customer registration
│   │   │   └── page.tsx
│   │   │
│   │   ├── catalogs/           # Catalog pages
│   │   │   └── [id]/
│   │   │       └── page.tsx    # Catalog detail
│   │   ├── hotels/
│   │   │   └── page.tsx        # Hotels listing
│   │   ├── transport/
│   │   │   └── page.tsx        # Transport listing
│   │   ├── food/
│   │   │   └── page.tsx        # Food plans listing
│   │   │
│   │   ├── profile/            # Customer profile (protected)
│   │   │   ├── page.tsx        # View profile
│   │   │   └── edit/
│   │   │       └── page.tsx    # Edit profile
│   │   │
│   │   ├── bookings/           # Customer bookings (protected)
│   │   │   ├── page.tsx        # Bookings list
│   │   │   ├── [id]/
│   │   │   │   ├── page.tsx    # Booking detail
│   │   │   │   ├── pay/
│   │   │   │   │   └── page.tsx # Payment flow
│   │   │   │   └── cancel/
│   │   │   │       └── page.tsx # Cancel booking
│   │   │   └── new/
│   │   │       ├── catalog/
│   │   │       │   └── [id]/
│   │   │       │       └── page.tsx # Book from catalog
│   │   │       └── custom/
│   │   │           └── page.tsx # Custom booking
│   │   │
│   │   └── admin/              # Admin pages (admin-only)
│   │       ├── login/
│   │       │   └── page.tsx
│   │       ├── dashboard/
│   │       │   └── page.tsx
│   │       ├── customers/
│   │       │   ├── page.tsx
│   │       │   └── [id]/
│   │       │       └── page.tsx
│   │       ├── bookings/
│   │       │   ├── page.tsx
│   │       │   └── [id]/
│   │       │       └── page.tsx
│   │       ├── payments/
│   │       │   ├── page.tsx
│   │       │   └── [id]/
│   │       │       └── page.tsx
│   │       ├── catalogs/
│   │       │   ├── page.tsx
│   │       │   ├── new/
│   │       │   │   └── page.tsx
│   │       │   └── [id]/
│   │       │       └── page.tsx
│   │       ├── hotels/
│   │       │   ├── page.tsx
│   │       │   └── [id]/
│   │       │       └── page.tsx
│   │       ├── transport/
│   │       │   ├── page.tsx
│   │       │   └── [id]/
│   │       │       └── page.tsx
│   │       └── food/
│   │           ├── page.tsx
│   │           └── [id]/
│   │               └── page.tsx
│   │
│   ├── components/
│   │   ├── ui/                 # shadcn/ui components
│   │   │   ├── button.tsx
│   │   │   ├── card.tsx
│   │   │   ├── input.tsx
│   │   │   ├── table.tsx
│   │   │   ├── badge.tsx
│   │   │   ├── dialog.tsx
│   │   │   ├── select.tsx
│   │   │   ├── skeleton.tsx
│   │   │   ├── calendar.tsx
│   │   │   ├── dropdown-menu.tsx
│   │   │   ├── avatar.tsx
│   │   │   ├── tabs.tsx
│   │   │   ├── separator.tsx
│   │   │   ├── alert.tsx
│   │   │   ├── alert-dialog.tsx
│   │   │   ├── form.tsx
│   │   │   ├── label.tsx
│   │   │   ├── textarea.tsx
│   │   │   └── sonner.tsx
│   │   │
│   │   ├── navbar.tsx          # Navigation bar
│   │   ├── protected-route.tsx # Customer route guard
│   │   └── admin-route.tsx     # Admin route guard
│   │
│   └── lib/
│       ├── api.ts              # Axios instance + API helpers
│       ├── auth-context.tsx    # Authentication context
│       └── utils.ts            # Utility functions (cn, etc.)
```

```
│
├── package.json
├── tsconfig.json
├── next.config.ts
├── postcss.config.mjs
├── tailwind.config.ts
└── components.json            # shadcn/ui configuration
```

---

5. Authentication

JWT Token Strategy

Storage: `localStorage` (keys: `token` , `user`)

Token Structure:

```
interface JWTPayload {
  id: number;         // customerId or adminId
  username: string;
  role: 'customer' | 'admin';
  iat: number;        // Issued at (Unix timestamp)
  exp: number;        // Expiration (Unix timestamp)
}
```

AuthContext API

```
interface AuthContextType {
  user: User | null;
  token: string | null;
  isLoading: boolean;
  isAuthenticated: boolean;
  isAdmin: boolean;
  login: (token: string, userData: Partial<User>) => void;
  logout: () => void;
}

interface User {
  id: number;
  username: string;
  role: 'customer' | 'admin';
  name?: string;
  email?: string;
}
```

Authentication Flow

```
1. User submits login form
2. POST to /api/auth/customer/login (or /api/auth/admin/login)
3. Backend returns { token, customer/admin }
```

```
4. Frontend decodes token, stores in localStorage
5. AuthContext updates state, triggers re-render
6. Protected routes become accessible
```

Token Expiration Handling

- On `AuthProvider` mount: Check token expiration
- If expired: Clear localStorage, set user to null
- On 401 response (interceptor): Clear storage, redirect to login

Route Guards

ProtectedRoute - For customer-only pages:

```
// Redirects to /login if not authenticated
<ProtectedRoute>
  <CustomerContent />
</ProtectedRoute>
```

AdminRoute - For admin-only pages:

```
// Redirects to /admin/login if not authenticated
// Redirects to / if authenticated but not admin
<AdminRoute>
  <AdminContent />
</AdminRoute>
```

---

6. Routing & Navigation

Route Map

Path	Access	Description
`/`	Public	Home - Catalog listing
`/catalogs/[id]`	Public	Catalog detail
`/hotels`	Public	Hotels listing
`/transport`	Public	Transport listing
`/food`	Public	Food plans listing
`/login`	Public	Customer login
`/register`	Public	Customer registration
`/profile`	Customer	View profile & summary
`/profile/edit`	Customer	Edit profile
`/bookings`	Customer	Customer's bookings list

`/bookings/[id]`	Customer	Booking detail
`/bookings/[id]/pay`	Customer	Payment flow
`/bookings/[id]/cancel`	Customer	Cancel booking
`/bookings/new/catalog/[id]`	Customer	Book from catalog
`/bookings/new/custom`	Customer	Create custom booking
`/admin/login`	Public	Admin login
`/admin/dashboard`	Admin	Dashboard & analytics
`/admin/customers`	Admin	Customers list
`/admin/customers/[id]`	Admin	Customer detail
`/admin/bookings`	Admin	All bookings
`/admin/bookings/[id]`	Admin	Booking detail + actions
`/admin/payments`	Admin	All payments
`/admin/payments/[id]`	Admin	Payment detail
`/admin/catalogs`	Admin	Catalogs management
`/admin/catalogs/new`	Admin	Create catalog
`/admin/catalogs/[id]`	Admin	Edit catalog
`/admin/hotels`	Admin	Hotels management
`/admin/hotels/[id]`	Admin	Hotel detail/edit
`/admin/transport`	Admin	Transport management
`/admin/transport/[id]`	Admin	Transport detail/edit
`/admin/food`	Admin	Food plans management
`/admin/food/[id]`	Admin	Food plan detail/edit

Navbar Configuration

Public/Customer Navigation:

```
const publicLinks = [
  { href: '/', label: 'Packages' },
  { href: '/hotels', label: 'Hotels' },
  { href: '/transport', label: 'Transport' },
  { href: '/food', label: 'Food' },
];
```

Admin Navigation:

```
const adminLinks = [
  { href: '/admin/dashboard', label: 'Dashboard' },
  { href: '/admin/bookings', label: 'Bookings' },
  { href: '/admin/payments', label: 'Payments' },
  { href: '/admin/customers', label: 'Customers' },
  { href: '/admin/catalogs', label: 'Catalogs' },
  { href: '/admin/hotels', label: 'Hotels' },
  { href: '/admin/transport', label: 'Transport' },
  { href: '/admin/food', label: 'Food' },
];
```

User Menu (Authenticated Customer):

- Profile
- My Bookings
- Settings
- Logout

---

7. Components

Layout Components

Root Layout (`layout.tsx`)

```
<html>
  <body>
    <AuthProvider>
      <Navbar />
      <main>{children}</main>
      <Toaster position="top-right" />
    </AuthProvider>
  </body>
</html>
```

Navbar

- Displays logo with "Travel Planner" branding
- Shows "Admin" badge when on admin routes
- Conditional navigation links based on route context
- User avatar dropdown when authenticated
- Login/Register buttons when not authenticated
- Mobile-responsive hamburger menu

Route Guard Components

ProtectedRoute

```
interface ProtectedRouteProps {
  children: React.ReactNode;
}
```

```
// Shows skeleton while loading
// Redirects to /login if not authenticated
// Renders children if authenticated
```

AdminRoute

```
interface AdminRouteProps {
  children: React.ReactNode;
}

// Shows skeleton while loading
// Redirects to /admin/login if not authenticated
// Redirects to / if authenticated but not admin
// Renders children if admin
```

UI Component Patterns

Card Pattern (Listing Items)

```
<Card className="flex flex-col">
  <CardHeader>
    <CardTitle>{title}</CardTitle>
    <CardDescription>{subtitle}</CardDescription>
  </CardHeader>
  <CardContent className="flex-1">
    {/* Main content */}
  </CardContent>
  <CardFooter>
    <Button asChild className="w-full">
      <Link href={detailUrl}>View Details</Link>
    </Button>
  </CardFooter>
</Card>
```

Stats Card Pattern (Dashboard)

```
<Card>
  <CardHeader className="flex flex-row items-center justify-between pb-2">
    <CardTitle className="text-sm font-medium">{label}</CardTitle>
    <Icon className="h-4 w-4 text-muted-foreground" />
  </CardHeader>
  <CardContent>
    <div className="text-2xl font-bold">{value}</div>
  </CardContent>
</Card>
```

Form Pattern

```
<form onSubmit={handleSubmit}>
  <CardContent className="space-y-4">
    <div className="space-y-2">
      <Label htmlFor="fieldName">Field Label</Label>
      <Input
        id="fieldName"
        name="fieldName"
        value={formData.fieldName}
        onChange={handleChange}
        required
        disabled={loading}
      />
    </div>
  </CardContent>
  <CardFooter>
    <Button type="submit" disabled={loading}>
      {loading && <Loader2 className="mr-2 h-4 w-4 animate-spin" />}
      Submit
    </Button>
  </CardFooter>
</form>
```

Table Pattern

```
<div className="rounded-md border">
  <Table>
    <TableHeader>
      <TableRow>
        <TableHead>Column 1</TableHead>
        <TableHead>Column 2</TableHead>
        <TableHead className="text-right">Actions</TableHead>
      </TableRow>
    </TableHeader>
    <TableBody>
      {items.map((item) => (
        <TableRow key={item.id}>
          <TableCell>{item.field1}</TableCell>
          <TableCell>{item.field2}</TableCell>
          <TableCell className="text-right">
            <Button variant="ghost" size="sm" asChild>
              <Link href={`/path/${item.id}`}>View</Link>
            </Button>
          </TableCell>
        </TableRow>
      ))}
    </TableBody>
  </Table>
</div>
```

Loading Skeleton Pattern

```
if (loading) {
  return (
    <div className="container mx-auto px-4 py-8">
      <Skeleton className="h-10 w-64 mb-2" />
      <Skeleton className="h-5 w-96" />
      <div className="grid gap-6 md:grid-cols-2 lg:grid-cols-3 mt-8">
        {[...Array(6)].map((_, i) => (
          <Card key={i}>
            <CardHeader>
              <Skeleton className="h-6 w-48 mb-2" />
              <Skeleton className="h-4 w-32" />
            </CardHeader>
            <CardContent>
              <Skeleton className="h-16 w-full" />
            </CardContent>
          </Card>
        ))}
      </div>
    </div>
  );
}
```

---

8. State Management

Client-Side State

Authentication State (Context)

```
// Managed by AuthProvider
const { user, token, isAuthenticated, isAdmin, login, logout } = useAuth();
```

Component-Level State (useState)

```
// Data fetching
const [data, setData] = useState<DataType[]>([]);
const [loading, setLoading] = useState(true);

// Form state
const [formData, setFormData] = useState({ field: '' });
const [processing, setProcessing] = useState(false);
```

State Patterns by Page Type

Listing Pages:

```
const [items, setItems] = useState<Item[]>([]);
const [loading, setLoading] = useState(true);

useEffect(() => {
  const fetchItems = async () => {
    try {
      const response = await itemApi.getAll();
      setItems(response.data);
    } catch (error) {
      toast.error('Failed to load items');
    } finally {
      setLoading(false);
    }
  };
  fetchItems();
}, []);
```

Detail Pages:

```
const params = useParams();
const [item, setItem] = useState<Item | null>(null);
const [loading, setLoading] = useState(true);

useEffect(() => {
  const fetchItem = async () => {
    try {
      const response = await itemApi.getById(params.id);
      setItem(response.data);
    } catch (error) {
      toast.error('Failed to load item');
    } finally {
      setLoading(false);
    }
  };
  if (params.id) fetchItem();
}, [params.id]);
```

Form Pages:

```
const [formData, setFormData] = useState(initialState);
const [loading, setLoading] = useState(false);

const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  setFormData(prev => ({ ...prev, [e.target.name]: e.target.value }));
};

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  setLoading(true);
  try {
    await itemApi.create(formData);
    toast.success('Created successfully');
    router.push('/items');
  } catch (error) {
    toast.error('Failed to create');
  } finally {
    setLoading(false);
  }
};
```

---

9. API Integration

Axios Instance Configuration

```
// src/lib/api.ts

const API_URL = process.env.NEXT_PUBLIC_API_URL || 'http://localhost:3001';

export const api = axios.create({
  baseURL: API_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});
```

Request Interceptor

```
api.interceptors.request.use(
  (config) => {
    if (typeof window !== 'undefined') {
      const token = localStorage.getItem('token');
      if (token) {
        config.headers.Authorization = `Bearer ${token}`;
      }
    }
    return config;
  },
  (error) => Promise.reject(error)
);
```

Response Interceptor

```
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      if (typeof window !== 'undefined') {
        localStorage.removeItem('token');
        localStorage.removeItem('user');
        if (!window.location.pathname.includes('/login')) {
          window.location.href = '/login';
        }
      }
    }
    return Promise.reject(error);
  }
);
```

API Helper Objects

authApi

```
export const authApi = {
  customerLogin: (username: string, password: string) =>
    api.post('/api/auth/customer/login', { username, password }),
  customerRegister: (data: CustomerRegistration) =>
    api.post('/api/auth/customer/register', data),
  adminLogin: (username: string, password: string) =>
    api.post('/api/auth/admin/login', { username, password }),
};
```

customerApi

```
export const customerApi = {
  getAll: () => api.get('/api/customers'),
  getById: (id: number) => api.get(`/api/customers/${id}`),
  getSummary: (id: number) => api.get(`/api/customers/${id}/summary`),
  update: (id: number, data: CustomerUpdate) => api.put(`/api/customers/${id}`, data),
  delete: (id: number) => api.delete(`/api/customers/${id}`),
};
```

catalogApi

```
export const catalogApi = {
  getAll: () => api.get('/api/catalogs'),
  getById: (id: number) => api.get(`/api/catalogs/${id}`),
  create: (data: CatalogCreate) => api.post('/api/catalogs', data),
  addHotel: (catalogId: number, data: CatalogHotel) =>
    api.post(`/api/catalogs/${catalogId}/hotels`, data),
  addTransport: (catalogId: number, data: CatalogTransport) =>
    api.post(`/api/catalogs/${catalogId}/transport`, data),
  addFood: (catalogId: number, data: CatalogFood) =>
    api.post(`/api/catalogs/${catalogId}/food`, data),
};
```

hotelApi

```
export const hotelApi = {
  getAll: () => api.get('/api/hotels'),
  getAvailability: () => api.get('/api/hotels/availability'),
  getById: (id: number) => api.get(`/api/hotels/${id}`),
  create: (data: HotelCreate) => api.post('/api/hotels', data),
  update: (id: number, data: HotelUpdate) => api.put(`/api/hotels/${id}`, data),
  delete: (id: number) => api.delete(`/api/hotels/${id}`),
};
```

transportApi

```
export const transportApi = {
  getAll: () => api.get('/api/transport'),
  getById: (id: number) => api.get(`/api/transport/${id}`),
  create: (data: TransportCreate) => api.post('/api/transport', data),
  update: (id: number, data: TransportUpdate) => api.put(`/api/transport/${id}`, data),
  delete: (id: number) => api.delete(`/api/transport/${id}`),
};
```

foodApi

```
export const foodApi = {
  getAll: () => api.get('/api/food'),
  getById: (id: number) => api.get(`/api/food/${id}`),
  create: (data: FoodCreate) => api.post('/api/food', data),
  update: (id: number, data: FoodUpdate) => api.put(`/api/food/${id}`, data),
  delete: (id: number) => api.delete(`/api/food/${id}`),
};
```

bookingApi

```
export const bookingApi = {
  getAll: () => api.get('/api/bookings'),
  getByCustomer: (customerId: number) => api.get(`/api/bookings/customer/${customerId}`),
  getById: (id: string) => api.get(`/api/bookings/${id}`),
  getCostBreakdown: (id: string) => api.get(`/api/bookings/${id}/total`),
  createFromCatalog: (data: CatalogBookingCreate) => api.post('/api/bookings/catalog',
data),
  createCustom: (data: CustomBookingCreate) => api.post('/api/bookings/custom', data),
  cancel: (id: string, reason: string) => api.post(`/api/bookings/${id}/cancel`, { reason
}),
  updateStatus: (id: string, status: string) => api.patch(`/api/bookings/${id}/status`, {
status }),
  delete: (id: string) => api.delete(`/api/bookings/${id}`),
};
```

paymentApi

```
export const paymentApi = {
  getAll: () => api.get('/api/payments'),
  getByBooking: (bookingId: string) => api.get(`/api/payments/booking/${bookingId}`),
  process: (data: PaymentProcess) => api.post('/api/payments/process', data),
  complete: (paymentId: string, transactionId: string) =>
    api.post(`/api/payments/${paymentId}/complete`, { transactionId }),
  updateStatus: (paymentId: string, data: PaymentStatusUpdate) =>
    api.patch(`/api/payments/${paymentId}/status`, data),
};
```

reportsApi

```
export const reportsApi = {
  getDashboard: () => api.get('/api/reports/dashboard'),
  getRevenue: () => api.get('/api/reports/revenue'),
};
```

---

10. Pages Specification

Public Pages

Home Page (`/`)

Purpose: Display all available travel packages

Data Source: `GET /api/catalogs`

UI Elements:

- Page title: "Travel Packages"
- Grid of catalog cards (3 columns on desktop)
- Each card shows: package name, destination, duration, price, services included
- "View Details" button per card

Empty State: "No travel packages available at the moment."

Catalog Detail (`/catalogs/[id]`)

Purpose: Show full package details

Data Source: `GET /api/catalogs/:id`

UI Elements:

- Package name and destination
- Description
- Travel dates (departure → arrival)
- Duration in days
- Hotels section with details
- Transport section with details
- Food plans section
- Total price display
- "Book This Package" button (links to booking form)

Hotels Page (`/hotels`)

Purpose: List all available hotels

Data Source: `GET /api/hotels`

UI Elements:

- Grid of hotel cards
- Each card: name, address, available rooms, price per night

Transport Page (`/transport`)

Purpose: List all transport options

Data Source: `GET /api/transport`

UI Elements:

- Grid of transport cards
- Each card: type, available seats, fare

Food Page (`/food`)

Purpose: List all meal plans

Data Source: `GET /api/food`

UI Elements:

- Grid of food plan cards
- Each card: meals description, price

Authentication Pages

Customer Login (`/login`)

Purpose: Authenticate customers

API: `POST /api/auth/customer/login`

Form Fields:

- Username (required)
- Password (required)

Success: Store token, redirect to `/` **Error:** Show toast error message

Links:

- Register link
- Admin login link

Customer Register (`/register`)

Purpose: Create new customer account

API: `POST /api/auth/customer/register`

Form Fields:

- Full Name (required)

- Email (required, email format)
- Phone (required)
- Username (required)
- Password (required, min 6 chars)
- Confirm Password (required, must match)

Success: Show success toast, redirect to `/login`

Admin Login (`/admin/login`)

Purpose: Authenticate administrators

API: `POST /api/auth/admin/login`

Form Fields:

- Username (required)
- Password (required)

Success: Store token, redirect to `/admin/dashboard`

Customer Pages (Protected)

Profile (`/profile`)

Purpose: View customer profile and booking summary

Data Source: `GET /api/customers/:id/summary`

UI Elements:

- Customer info card (name, email, phone, username)
- Stats cards: total bookings, confirmed, cancelled
- Total spent
- Customer tier badge
- Edit profile button

Edit Profile (`/profile/edit`)

Purpose: Update customer information

API: `PUT /api/customers/:id`

Form Fields:

- Name
- Email
- Phone
- New Password (optional)
- Confirm Password

My Bookings (`/bookings`)

Purpose: List customer's bookings

Data Source: `GET /api/bookings/customer/:customerId`

UI Elements:

- Table of bookings
- Columns: ID, Package/Description, Date, Status, Total, Payment Status
- Status badges (confirmed/pending/cancelled)
- View button per row

Booking Detail (`/bookings/[id]`)

Purpose: Full booking information

Data Source: `GET /api/bookings/:id` , `GET /api/bookings/:id/total`

UI Elements:

- Booking summary card
- Hotels, transport, food details (if applicable)
- Cost breakdown
- Status and payment status badges
- Action buttons:
  - "Pay Now" (if unpaid)
  - "Cancel Booking" (if not cancelled)

Book from Catalog (`/bookings/new/catalog/[id]`)

Purpose: Create booking from catalog package

API: `POST /api/bookings/catalog`

Form Fields:

- Check-in date (date picker)
- Check-out date (date picker)
- Travel date (date picker)
- Description/notes

Display: Package summary with pricing

Custom Booking (`/bookings/new/custom`)

Purpose: Build custom trip

API: `POST /api/bookings/custom`

UI Elements:

- Description field
- Hotels selection (multi-select with rooms, dates)
- Transport selection (multi-select with seats, date)
- Food selection (multi-select with quantity)
- Running total calculation
- Submit button

Payment Page (`/bookings/[id]/pay`)

Purpose: Two-step payment flow

APIs: `POST /api/payments/process` , `POST /api/payments/:id/complete`

Step 1 - Process:

- Payment method dropdown
- Transaction ID (optional, auto-generated)
- "Initiate Payment" button

Step 2 - Complete:

- Payment summary display
- Confirmation transaction ID
- "Complete Payment" button

Done State: Success message with link to booking

Cancel Booking (`/bookings/[id]/cancel`)

Purpose: Cancel a booking

API: `POST /api/bookings/:id/cancel`

Form Fields:

- Cancellation reason (textarea)
- Confirmation dialog

Success: Redirect to bookings list

Admin Pages (Admin-Only)

Dashboard (`/admin/dashboard`)

Purpose: Business overview

Data Sources: `GET /api/reports/dashboard` , `GET /api/reports/revenue`

UI Elements:

- Stats cards: Total Customers, Total Bookings, Confirmed, Pending, Cancelled, Total Revenue, Total Catalogs
- Monthly revenue table

Customers Management (`/admin/customers`)

Purpose: View all customers

Data Source: `GET /api/customers`

UI Elements:

- Table: ID, Name, Email, Phone, Username
- View button per row

Customer Detail (`/admin/customers/[id]`)

Purpose: View customer with booking summary

Data Source: `GET /api/customers/:id/summary`

UI Elements:

- Customer info
- Booking statistics
- Delete customer action (with confirmation)

Bookings Management (`/admin/bookings`)

Purpose: View all bookings

Data Source: `GET /api/bookings`

UI Elements:

- Table: ID, Customer, Package, Date, Status, Total, Payment
- Filter options
- View button per row

Admin Booking Detail (`/admin/bookings/[id]`)

Purpose: Full booking management

UI Elements:

- All booking details
- Status update dropdown
- Payment status
- Delete booking action

Payments Management (`/admin/payments`)

Purpose: View all payments

Data Source: `GET /api/payments`

UI Elements:

- Table: ID, Booking ID, Customer, Amount, Date, Method, Status
- View button per row

Payment Detail (`/admin/payments/[id]`)

Purpose: View and update payment

UI Elements:

- Payment details
- Status update dropdown

Catalogs Management (`/admin/catalogs`)

Purpose: Manage travel packages

UI Elements:

- Table of catalogs
- "Create New" button

- Edit button per row

Create/Edit Catalog (`/admin/catalogs/new` , `/admin/catalogs/[id]`)

Purpose: Create or edit catalog

Form Fields:

- Package Name
- Destination
- Description
- Number of Days
- Budget
- Departure Date
- Arrival Date

Associated Items (Edit only):

- Add hotels with rooms
- Add transport with seats
- Add food plans

Hotels/Transport/Food Management

CRUD Pages for each service type:

- Listing page with table
- Detail/Edit page with form
- Create functionality
- Delete with confirmation

---

11. TypeScript Interfaces

Core Entities

```typescript
// Customer
interface Customer {
  customerId: number;
  name: string;
  email: string;
  phone: string;
  username: string;
}

interface CustomerSummary extends Customer {
  totalBookings: number;
  confirmedBookings: number;
  cancelledBookings: number;
  totalSpent: number;
  customerTier: string;
}
```

```typescript
// Catalog
interface Catalog {
  catalogId: number;
  packageName: string;
  destination: string;
  description: string;
  noOfDays: number;
  budget: number;
  departure: string;
  arrival: string;
  totalHotels: number;
  totalHotelCost: number;
  totalTransports: number;
  totalTransportCost: number;
  totalFoodPlans: number;
  totalFoodCost: number;
  calculatedTotalCost: number;
}

interface CatalogDetail extends Catalog {
  hotels: CatalogHotel[];
  transport: CatalogTransport[];
  food: CatalogFood[];
}

// Hotel
interface Hotel {
  hotelId: number;
  hotelName: string;
  hotelAddress: string;
  availableRooms: number;
  rent: number;
}

interface HotelAvailability extends Hotel {
  totalRooms: number;
  pricePerNight: number;
  currentlyBookedRooms: number;
  availableRoomsNow: number;
}

// Transport
interface Transport {
  transportId: number;
  type: string;
  availableSeats: number;
  fare: number;
}

// Food
interface Food {
  foodId: number;
  meals: string;
  price: number;
}

// Booking
interface Booking {
  bookingId: string;
  customerId: number;
  customerName: string;
  customerEmail: string;
  catalogId: number | null;
  packageName: string | null;
  destination: string | null;
  isCustom: boolean;
  bookingType: 'package' | 'custom';
  bookingDescription: string;
  bookingDate: string;
  status: 'pending' | 'confirmed' | 'cancelled';
  totalHotelCost: number;
  totalTransportCost: number;
  totalFoodCost: number;
  grandTotal: number;
  paymentStatus: 'unpaid' | 'pending' | 'completed' | 'refunded';
  paidAmount: number;
}

interface BookingCostBreakdown {
  hotelCost: number;
  transportCost: number;
  foodCost: number;
  totalCost: number;
}

// Payment
interface Payment {
  paymentId: string;
  bookingId: string;
  amount: number;
  paymentDate: string;
  method: string;
  status: 'pending' | 'completed' | 'refunded';
  transactionId: string;
  customerName?: string;
}

// Dashboard Stats
interface DashboardStats {
  totalCustomers: number;
  totalBookings: number;
  confirmedBookings: number;
  pendingBookings: number;
  cancelledBookings: number;
  totalRevenue: number;
  totalCatalogs: number;
}

// Revenue Report
interface RevenueReport {
  paymentMonth: string;
  totalTransactions: number;
  totalBookings: number;
  uniqueCustomers: number;
  completedRevenue: number;
  pendingRevenue: number;
  refundedAmount: number;
  packageRevenue: number;
  customBookingRevenue: number;
}
```

Form/Request Types

```
// Auth
interface CustomerRegistration {
  name: string;
  email: string;
  phone: string;
  username: string;
  password: string;
}

interface LoginCredentials {
  username: string;
  password: string;
}

// Booking Creation
interface CatalogBookingCreate {
  customerId: number;
  catalogId: number;
  bookingDescription: string;
  checkIn: string;
  checkOut: string;
  travelDate: string;
}

interface CustomBookingCreate {
  customerId: number;
  bookingDescription: string;
  hotels?: {
    hotelId: number;
    roomsBooked: number;
    checkIn: string;
    checkOut: string;
  }[];
  transport?: {
    transportId: number;
    seatsBooked: number;
    travelDate: string;
  }[];
  food?: {
    foodId: number;
    quantity: number;
  }[];
}

// Payment
interface PaymentProcess {
  bookingId: string;
  method: string;
  transactionId: string;
}

interface PaymentStatusUpdate {
  status: string;
  transactionId?: string;
}
```

---

12. Styling & UI Guidelines

Tailwind Configuration

Uses Tailwind CSS v4 with CSS variables for theming.

Color Scheme

```
/* Light Mode */
--background: 0 0% 100%;
--foreground: 0 0% 3.9%;
--primary: 0 0% 9%;
--primary-foreground: 0 0% 98%;
--muted: 0 0% 96.1%;
--muted-foreground: 0 0% 45.1%;
--accent: 0 0% 96.1%;
--destructive: 0 84.2% 60.2%;
```

Spacing Standards

Element	Spacing
Container padding	`px-4`
Page top padding	`py-8`

Card gaps	`gap-6`
Form field gaps	`space-y-4`
Section margins	`mb-8`

Responsive Breakpoints

Breakpoint	Usage
`md:`	768px - Tablet
`lg:`	1024px - Desktop

Grid Layouts

```
// Card grids
<div className="grid gap-6 md:grid-cols-2 lg:grid-cols-3">

// Stats grids
<div className="grid gap-4 md:grid-cols-2 lg:grid-cols-4">
```

Status Badge Colors

```
// Booking Status
<Badge variant={status === 'confirmed' ? 'default' :
               status === 'pending' ? 'secondary' :
               'destructive'}>
  {status}
</Badge>

// Payment Status
<Badge variant={status === 'completed' ? 'default' :
               status === 'pending' ? 'secondary' :
               'outline'}>
  {status}
</Badge>
```

Button Variants

Variant	Usage
`default`	Primary actions
`secondary`	Secondary actions
`outline`	Tertiary actions
`ghost`	Navigation, icon buttons
`destructive`	Delete, cancel actions

`link`	Text links

---

13. Error Handling

API Error Pattern

```
try {
  const response = await api.someAction();
  // Handle success
} catch (error: unknown) {
  console.error('Action failed:', error);
  const err = error as { response?: { data?: { message?: string } } };
  toast.error(err.response?.data?.message || 'Default error message');
} finally {
  setLoading(false);
}
```

Error Display Methods

Toast Notifications (Sonner):

```
// Success
toast.success('Action completed successfully');

// Error
toast.error('Something went wrong');

// Info
toast.info('Please note...');
```

Empty States:

```
{items.length === 0 && (
  <div className="text-center py-12">
    <p className="text-muted-foreground text-lg">No items found</p>
  </div>
)}
```

Not Found:

```
if (!item) {
  return (
    <div className="text-center py-12">
      <p className="text-muted-foreground text-lg mb-4">Item not found</p>
      <Button asChild variant="outline">
        <Link href="/items">
          <ArrowLeft className="mr-2 h-4 w-4" />
          Back to List
        </Link>
      </Button>
    </div>
  );
}
```

HTTP Status Code Handling

Status	Action
401	Clear auth, redirect to login
403	Show forbidden message
404	Show not found state
400	Show validation error from response
500	Show generic server error

---

14. Environment Variables

Required Variables

Variable	Default	Description
`NEXT_PUBLIC_API_URL`	`http://localhost:3001`	Backend API base URL

Configuration Files

.env.local (Development):

```
NEXT_PUBLIC_API_URL=http://localhost:3001
```

.env.production:

```
NEXT_PUBLIC_API_URL=https://api.yourproduction.com
```

Appendix A: Page Count Summary

Category	Pages
Public	7
Authentication	3
Customer	8
Admin	19
Total	**37**

Appendix B: Component Count

Type	Count
UI Components (shadcn)	19
Layout Components	2
Route Guards	2
Total	**23**

Appendix C: API Endpoint Summary (Frontend Usage)

Category	Endpoints
Auth	3
Customers	5
Catalogs	6
Hotels	6
Transport	5
Food	5
Bookings	9
Payments	5
Reports	2
Total	**46**