**Especificaciones Técnicas de Implementación - TurisGal**

**1. STACK TECNOLÓGICO DETALLADO**

**Frontend Móvil (React Native)**

**Dependencias Principales**

{

"dependencies": {

"react": "18.2.0",

"react-native": "0.72.6",

"expo": "~49.0.15",

"@react-navigation/native": "^6.1.9",

"@react-navigation/stack": "^6.3.20",

"@react-navigation/bottom-tabs": "^6.5.11",

"expo-camera": "~13.6.0",

"expo-barcode-scanner": "~12.6.0",

"expo-image-picker": "~14.5.2",

"expo-location": "~16.3.0",

"expo-notifications": "~0.23.0",

"@react-native-async-storage/async-storage": "1.19.3",

"react-native-qrcode-scanner": "^1.5.5",

"react-native-signature-canvas": "^4.6.1",

"react-native-image-resizer": "^3.0.4",

"@reduxjs/toolkit": "^1.9.7",

"react-redux": "^8.1.3",

"axios": "^1.6.0",

"react-native-paper": "^5.11.1",

"react-native-vector-icons": "^10.0.2",

"react-hook-form": "^7.47.0",

"yup": "^1.3.3",

"react-native-reanimated": "~3.5.4",

"react-native-gesture-handler": "~2.12.0",

"expo-secure-store": "~12.5.0",

"react-native-super-grid": "^4.4.4",

"react-native-calendars": "^1.1302.0"

},

"devDependencies": {

"@types/react": "~18.2.14",

"@types/react-native": "~0.72.2",

"typescript": "^5.1.3",

"eslint": "^8.51.0",

"prettier": "^3.0.3",

"@testing-library/react-native": "^12.3.2",

"jest": "^29.2.1"

}

}

**Estructura de Carpetas React Native**

src/

├── components/ # Componentes reutilizables

│ ├── ui/ # Componentes básicos (Button, Input, etc.)

│ ├── forms/ # Componentes de formularios

│ ├── camera/ # Componentes relacionados con cámara

│ └── navigation/ # Componentes de navegación

├── screens/ # Pantallas principales

│ ├── auth/ # Autenticación

│ ├── dashboard/ # Dashboard principal

│ ├── checkin/ # Proceso de check-in

│ ├── checkout/ # Proceso de check-out

│ ├── bookings/ # Gestión de reservas

│ ├── reviews/ # Sistema de reseñas

│ └── profile/ # Perfil de usuario

├── navigation/ # Configuración de navegación

├── store/ # Redux store y slices

├── services/ # Servicios API y utilidades

├── hooks/ # Custom hooks

├── utils/ # Funciones utilitarias

├── constants/ # Constantes y configuración

├── types/ # TypeScript types

└── assets/ # Imágenes, iconos, etc.

**Configuración Redux Store**

// store/index.ts

import { configureStore } from '@reduxjs/toolkit';

import authSlice from './slices/authSlice';

import bookingsSlice from './slices/bookingsSlice';

import checkinSlice from './slices/checkinSlice';

import notificationsSlice from './slices/notificationsSlice';

export const store = configureStore({

reducer: {

auth: authSlice,

bookings: bookingsSlice,

checkin: checkinSlice,

notifications: notificationsSlice,

},

middleware: (getDefaultMiddleware) =>

getDefaultMiddleware({

serializableCheck: {

ignoredActions: ['persist/PERSIST'],

},

}),

});

export type RootState = ReturnType<typeof store.getState>;

export type AppDispatch = typeof store.dispatch;

**Slice de Autenticación**

// store/slices/authSlice.ts

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

import { authService } from '../../services/authService';

interface AuthState {

user: User | null;

token: string | null;

isLoading: boolean;

error: string | null;

isAuthenticated: boolean;

}

export const loginUser = createAsyncThunk(

'auth/loginUser',

async (credentials: LoginCredentials, { rejectWithValue }) => {

try {

const response = await authService.login(credentials);

return response.data;

} catch (error: any) {

return rejectWithValue(error.response.data.message);

}

}

);

const authSlice = createSlice({

name: 'auth',

initialState: {

user: null,

token: null,

isLoading: false,

error: null,

isAuthenticated: false,

} as AuthState,

reducers: {

clearError: (state) => {

state.error = null;

},

logout: (state) => {

state.user = null;

state.token = null;

state.isAuthenticated = false;

},

},

extraReducers: (builder) => {

builder

.addCase(loginUser.pending, (state) => {

state.isLoading = true;

state.error = null;

})

.addCase(loginUser.fulfilled, (state, action) => {

state.isLoading = false;

state.user = action.payload.user;

state.token = action.payload.token;

state.isAuthenticated = true;

})

.addCase(loginUser.rejected, (state, action) => {

state.isLoading = false;

state.error = action.payload as string;

});

},

});

export const { clearError, logout } = authSlice.actions;

export default authSlice.reducer;

**Backend API (Node.js + Express)**

**Estructura del Proyecto Backend**

server/

├── src/

│ ├── controllers/ # Controladores de rutas

│ ├── middleware/ # Middleware personalizado

│ ├── models/ # Modelos de base de datos (Prisma)

│ ├── routes/ # Definición de rutas

│ ├── services/ # Lógica de negocio

│ ├── utils/ # Utilidades y helpers

│ ├── config/ # Configuración de la aplicación

│ ├── validators/ # Validadores de entrada

│ └── types/ # TypeScript types

├── prisma/ # Esquemas de base de datos

│ ├── schema.prisma # Definición del esquema

│ ├── migrations/ # Migraciones de BD

│ └── seed.ts # Datos de prueba

├── tests/ # Tests unitarios e integración

├── uploads/ # Archivos temporales

├── logs/ # Archivos de log

└── docs/ # Documentación API

**Package.json Backend**

{

"name": "turisgal-api",

"version": "1.0.0",

"dependencies": {

"express": "^4.18.2",

"cors": "^2.8.5",

"helmet": "^7.1.0",

"morgan": "^1.10.0",

"compression": "^1.7.4",

"express-rate-limit": "^7.1.5",

"express-validator": "^7.0.1",

"bcryptjs": "^2.4.3",

"jsonwebtoken": "^9.0.2",

"prisma": "^5.6.0",

"@prisma/client": "^5.6.0",

"multer": "^1.4.5-lts.1",

"aws-sdk": "^2.1489.0",

"nodemailer": "^6.9.7",

"twilio": "^4.19.0",

"qrcode": "^1.5.3",

"jimp": "^0.22.10",

"winston": "^3.11.0",

"dotenv": "^16.3.1",

"joi": "^17.11.0",

"socket.io": "^4.7.4",

"redis": "^4.6.10",

"bull": "^4.12.2",

"sharp": "^0.32.6",

"uuid": "^9.0.1"

},

"devDependencies": {

"@types/express": "^4.17.21",

"@types/node": "^20.8.10",

"@types/bcryptjs": "^2.4.6",

"@types/jsonwebtoken": "^9.0.5",

"@types/multer": "^1.4.11",

"typescript": "^5.2.2",

"ts-node": "^10.9.1",

"nodemon": "^3.0.1",

"jest": "^29.7.0",

"supertest": "^6.3.3",

"@types/jest": "^29.5.7"

}

}

**Configuración de Prisma Schema**

// prisma/schema.prisma

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "postgresql"

url = env("DATABASE\_URL")

}

model User {

id String @id @default(uuid())

email String @unique

phone String?

passwordHash String @map("password\_hash")

firstName String @map("first\_name")

lastName String @map("last\_name")

dateOfBirth DateTime? @map("date\_of\_birth")

nationality String?

profileImageUrl String? @map("profile\_image\_url")

preferredLanguage String @default("es") @map("preferred\_language")

isVerified Boolean @default(false) @map("is\_verified")

createdAt DateTime @default(now()) @map("created\_at")

updatedAt DateTime @updatedAt @map("updated\_at")

bookings Booking[]

checkIns CheckIn[]

reviews Review[]

notifications Notification[]

deviceTokens DeviceToken[]

@@map("users")

}

model PropertyOwner {

id String @id @default(uuid())

email String @unique

passwordHash String @map("password\_hash")

companyName String? @map("company\_name")

contactName String @map("contact\_name")

phone String?

taxId String? @map("tax\_id")

role String @default("owner")

permissions Json?

createdAt DateTime @default(now()) @map("created\_at")

properties Property[]

checkIns CheckIn[]

checkOuts CheckOut[]

notifications Notification[]

deviceTokens DeviceToken[]

@@map("property\_owners")

}

model Property {

id String @id @default(uuid())

ownerId String @map("owner\_id")

name String

description String?

propertyType String @map("property\_type")

address Json

totalRooms Int @default(1) @map("total\_rooms")

maxGuests Int @map("max\_guests")

amenities Json?

houseRules String? @map("house\_rules")

checkInTime String @default("15:00") @map("check\_in\_time")

checkOutTime String @default("11:00") @map("check\_out\_time")

qrCodeData String @unique @map("qr\_code\_data")

images Json?

isActive Boolean @default(true) @map("is\_active")

createdAt DateTime @default(now()) @map("created\_at")

owner PropertyOwner @relation(fields: [ownerId], references: [id])

rooms Room[]

bookings Booking[]

checkIns CheckIn[]

reviews Review[]

@@map("properties")

}

model Room {

id String @id @default(uuid())

propertyId String @map("property\_id")

roomNumber String @map("room\_number")

roomType String? @map("room\_type")

maxGuests Int @default(2) @map("max\_guests")

pricePerNight Decimal @map("price\_per\_night")

qrCodeData String @unique @map("qr\_code\_data")

isAvailable Boolean @default(true) @map("is\_available")

createdAt DateTime @default(now()) @map("created\_at")

property Property @relation(fields: [propertyId], references: [id])

bookings Booking[]

@@unique([propertyId, roomNumber])

@@map("rooms")

}

model Booking {

id String @id @default(uuid())

userId String @map("user\_id")

propertyId String @map("property\_id")

roomId String? @map("room\_id")

bookingReference String @unique @map("booking\_reference")

checkInDate DateTime @map("check\_in\_date")

checkOutDate DateTime @map("check\_out\_date")

guestsCount Int @map("guests\_count")

guestDetails Json? @map("guest\_details")

totalAmount Decimal? @map("total\_amount")

bookingStatus String @default("confirmed") @map("booking\_status")

specialRequests String? @map("special\_requests")

createdAt DateTime @default(now()) @map("created\_at")

user User @relation(fields: [userId], references: [id])

property Property @relation(fields: [propertyId], references: [id])

room Room? @relation(fields: [roomId], references: [id])

checkIns CheckIn[]

checkOuts CheckOut[]

reviews Review[]

@@map("bookings")

}

model CheckIn {

id String @id @default(uuid())

bookingId String @map("booking\_id")

userId String @map("user\_id")

propertyId String @map("property\_id")

checkInTimestamp DateTime @default(now()) @map("check\_in\_timestamp")

identityDocumentUrl String? @map("identity\_document\_url")

selfieUrl String? @map("selfie\_url")

digitalSignature String? @map("digital\_signature")

deviceInfo Json? @map("device\_info")

locationCoordinates String? @map("location\_coordinates")

verificationStatus String @default("pending") @map("verification\_status")

verifiedBy String? @map("verified\_by")

notes String?

booking Booking @relation(fields: [bookingId], references: [id])

user User @relation(fields: [userId], references: [id])

property Property @relation(fields: [propertyId], references: [id])

verifier PropertyOwner? @relation(fields: [verifiedBy], references: [id])

checkOuts CheckOut[]

@@map("check\_ins")

}

model CheckOut {

id String @id @default(uuid())

bookingId String @map("booking\_id")

checkInId String @map("check\_in\_id")

checkOutTimestamp DateTime @default(now()) @map("check\_out\_timestamp")

roomConditionPhotos Json? @map("room\_condition\_photos")

damagesReported String? @map("damages\_reported")

additionalCharges Decimal @default(0) @map("additional\_charges")

guestSignature String? @map("guest\_signature")

staffNotes String? @map("staff\_notes")

processedBy String? @map("processed\_by")

booking Booking @relation(fields: [bookingId], references: [id])

checkIn CheckIn @relation(fields: [checkInId], references: [id])

processor PropertyOwner? @relation(fields: [processedBy], references: [id])

@@map("check\_outs")

}

model Review {

id String @id @default(uuid())

bookingId String @map("booking\_id")

userId String @map("user\_id")

propertyId String @map("property\_id")

overallRating Int @map("overall\_rating")

cleanlinessRating Int @map("cleanliness\_rating")

locationRating Int @map("location\_rating")

valueRating Int @map("value\_rating")

serviceRating Int @map("service\_rating")

comment String?

photos Json?

isAnonymous Boolean @default(false) @map("is\_anonymous")

responseFromOwner String? @map("response\_from\_owner")

responseDate DateTime? @map("response\_date")

createdAt DateTime @default(now()) @map("created\_at")

booking Booking @relation(fields: [bookingId], references: [id])

user User @relation(fields: [userId], references: [id])

property Property @relation(fields: [propertyId], references: [id])

@@map("reviews")

}

model Notification {

id String @id @default(uuid())

userId String? @map("user\_id")

propertyOwnerId String? @map("property\_owner\_id")

notificationType String @map("notification\_type")

title String

message String

data Json?

isRead Boolean @default(false) @map("is\_read")

sentAt DateTime @default(now()) @map("sent\_at")

user User? @relation(fields: [userId], references: [id])

propertyOwner PropertyOwner? @relation(fields: [propertyOwnerId], references: [id])

@@map("notifications")

}

model DeviceToken {

id String @id @default(uuid())

userId String? @map("user\_id")

propertyOwnerId String? @map("property\_owner\_id")

token String @unique

platform String

isActive Boolean @default(true) @map("is\_active")

createdAt DateTime @default(now()) @map("created\_at")

user User? @relation(fields: [userId], references: [id])

propertyOwner PropertyOwner? @relation(fields: [propertyOwnerId], references: [id])

@@map("device\_tokens")

}

**Controlador de Check-in**

// src/controllers/checkinController.ts

import { Request, Response } from 'express';

import { checkinService } from '../services/checkinService';

import { validateQRCode } from '../validators/checkinValidator';

import { uploadService } from '../services/uploadService';

export class CheckinController {

async scanQR(req: Request, res: Response) {

try {

const { qrData } = req.body;

const validation = await validateQRCode(qrData);

if (!validation.isValid) {

return res.status(400).json({

success: false,

message: validation.error

});

}

const booking = await checkinService.getBookingByQR(qrData);

if (!booking) {

return res.status(404).json({

success: false,

message: 'Reserva no encontrada o código QR inválido'

});

}

// Verificar que el usuario actual puede hacer check-in

if (booking.userId !== req.user.id) {

return res.status(403).json({

success: false,

message: 'No autorizado para esta reserva'

});

}

// Verificar fechas

const today = new Date();

const checkInDate = new Date(booking.checkInDate);

if (today < checkInDate) {

return res.status(400).json({

success: false,

message: 'Aún no es la fecha de check-in'

});

}

res.json({

success: true,

data: {

booking: {

id: booking.id,

reference: booking.bookingReference,

property: booking.property,

room: booking.room,

checkInDate: booking.checkInDate,

checkOutDate: booking.checkOutDate,

guestsCount: booking.guestsCount

}

}

});

} catch (error) {

console.error('Error en scanQR:', error);

res.status(500).json({

success: false,

message: 'Error interno del servidor'

});

}

}

async uploadIdentityDocument(req: Request, res: Response) {

try {

const { bookingId } = req.params;

const files = req.files as { [fieldname: string]: Express.Multer.File[] };

if (!files.document || !files.selfie) {

return res.status(400).json({

success: false,

message: 'Se requieren tanto el documento como la selfie'

});

}

const documentFile = files.document[0];

const selfieFile = files.selfie[0];

// Validar archivos

const validation = await checkinService.validateIdentityFiles(

documentFile,

selfieFile

);

if (!validation.isValid) {

return res.status(400).json({

success: false,

message: validation.error

});

}

// Subir archivos a S3

const documentUrl = await uploadService.uploadFile(

documentFile,

'documents',

req.user.id

);

const selfieUrl = await uploadService.uploadFile(

selfieFile,

'selfies',

req.user.id

);

// Procesar verificación de identidad (OCR + face matching)

const verificationResult = await checkinService.processIdentityVerification(

documentUrl,

selfieUrl,

req.user.id

);

// Crear o actualizar check-in

const checkIn = await checkinService.updateCheckInWithIdentity(

bookingId,

{

identityDocumentUrl: documentUrl,

selfieUrl: selfieUrl,

verificationStatus: verificationResult.status,

deviceInfo: {

userAgent: req.headers['user-agent'],

ip: req.ip,

timestamp: new Date()

}

}

);

res.json({

success: true,

data: {

checkInId: checkIn.id,

verificationStatus: verificationResult.status,

nextStep: verificationResult.status === 'verified' ? 'terms' : 'manual\_review'

}

});

} catch (error) {

console.error('Error en uploadIdentityDocument:', error);

res.status(500).json({

success: false,

message: 'Error procesando documentos'

});

}

}

async completeCheckIn(req: Request, res: Response) {

try {

const { bookingId } = req.params;

const { digitalSignature, locationCoordinates } = req.body;

const checkIn = await checkinService.completeCheckIn(bookingId, {

digitalSignature,

locationCoordinates,

userId: req.user.id

});

// Actualizar estado de la reserva

await checkinService.updateBookingStatus(bookingId, 'checked\_in');

// Programar notificaciones futuras

await checkinService.scheduleCheckOutReminder(bookingId);

// Enviar notificación al propietario

await checkinService.notifyPropertyOwner(checkIn.propertyId, {

type: 'check\_in\_completed',

guestName: `${req.user.firstName} ${req.user.lastName}`,

property: checkIn.property.name

});

res.json({

success: true,

data: {

checkIn: {

id: checkIn.id,

timestamp: checkIn.checkInTimestamp,

property: checkIn.property,

accessInfo: {

wifi: checkIn.property.wifiCredentials,

instructions: checkIn.property.accessInstructions

}

}

}

});

} catch (error) {

console.error('Error en completeCheckIn:', error);

res.status(500).json({

success: false,

message: 'Error completando check-in'

});

}

}

}

export const checkinController = new CheckinController();

**Servicio de Check-in**

// src/services/checkinService.ts

import { PrismaClient } from '@prisma/client';

import { uploadService } from './uploadService';

import { ocrService } from './ocrService';

import { faceMatchingService } from './faceMatchingService';

import { notificationService } from './notificationService';

import { queueService } from './queueService';

const prisma = new PrismaClient();

export class CheckinService {

async getBookingByQR(qrData: string) {

return await prisma.booking.findFirst({

where: {

OR: [

{ property: { qrCodeData: qrData } },

{ room: { qrCodeData: qrData } }

],

bookingStatus: 'confirmed'

},

include: {

user: true,

property: true,

room: true

}

});

}

async validateIdentityFiles(documentFile: Express.Multer.File, selfieFile: Express.Multer.File) {

const maxSize = 10 \* 1024 \* 1024; // 10MB

const allowedTypes = ['image/jpeg', 'image/png', 'image/jpg'];

if (documentFile.size > maxSize || selfieFile.size > maxSize) {

return { isValid: false, error: 'Los archivos no pueden superar 10MB' };

}

if (!allowedTypes.includes(documentFile.mimetype) || !allowedTypes.includes(selfieFile.mimetype)) {

return { isValid: false, error: 'Solo se permiten archivos JPG, JPEG y PNG' };

}

return { isValid: true };

}

async processIdentityVerification(documentUrl: string, selfieUrl: string, userId: string) {

try {

// Procesar OCR del documento

const ocrResult = await ocrService.extractDocumentData(documentUrl);

if (!ocrResult.success) {

return {

status: 'failed',

reason: 'No se pudo extraer información del documento'

};

}

// Comparar rostros

const faceMatchResult = await faceMatchingService.compareImages(

documentUrl,

selfieUrl

);

if (faceMatchResult.confidence < 0.8) {

return {

status: 'manual\_review',

reason: 'Baja confianza en la comparación facial'

};

}

// Validar datos extraídos

const user = await prisma.user.findUnique({ where: { id: userId } });

const dataMatch = this.validateExtractedData(ocrResult.data, user);

if (!dataMatch.isValid) {

return {

status: 'manual\_review',

reason: dataMatch.reason

};

}

return {

status: 'verified',

extractedData: ocrResult.data,

confidence: faceMatchResult.confidence

};

} catch (error) {

console.error('Error en verificación de identidad:', error);

return {

status: 'failed',

reason: 'Error técnico en la verificación'

};

}

}

private validateExtractedData(extractedData: any, user: any) {

// Comparar nombres (tolerancia a diferencias menores)

const nameMatch = this.fuzzyMatch(

`${extractedData.firstName} ${extractedData.lastName}`,

`${user.firstName} ${user.lastName}`

);

if (nameMatch < 0.8) {

return {

isValid: false,

reason: 'Los nombres no coinciden suficientemente'

};

}

// Validar fecha de nacimiento si está disponible

if (extractedData.dateOfBirth && user.dateOfBirth) {

const docDate = new Date(extractedData.dateOfBirth);

const userDate = new Date(user.dateOfBirth);

if (Math.abs(docDate.getTime() - userDate.getTime()) > 24 \* 60 \* 60 \* 1000) {

return {

isValid: false,

reason: 'La fecha de nacimiento no coincide'

};

}

}

return { isValid: true };

}

private fuzzyMatch(str1: string, str2: string): number {

// Implementación simple de distancia de Levenshtein normalizada

const longer = str1.length > str2.length ? str1 : str2;

const shorter = str1.length > str2.length ? str2 : str1;

if (longer.length === 0) return 1.0;

const distance = this.levenshteinDistance(longer, shorter);

return (longer.length - distance) / longer.length;

}

private levenshteinDistance(str1: string, str2: string): number {

const matrix = [];

for (let i = 0; i <= str2.length; i++) {

matrix[i] = [i];

}

for (let j = 0; j <= str1.length; j++) {

matrix[0][j] = j;

}

for (let i = 1; i <= str2.length; i++) {

for (let j = 1; j <= str1.length; j++) {

if (str2.charAt(i - 1) === str1.charAt(j - 1)) {

matrix[i][j] = matrix[i - 1][j - 1];

} else {

matrix[i][j] = Math.min(

matrix[i - 1][j - 1] + 1,

matrix[i][j - 1] + 1,

matrix[i - 1][j] + 1

);

}

}

}

return matrix[str2.length][str1.length];

}

async updateCheckInWithIdentity(bookingId: string, data: any) {

return await prisma.checkIn.upsert({

where: {

bookingId: bookingId

},

update: data,

create: {

bookingId,

userId: data.userId,

propertyId: data.propertyId,

...data

},

include: {

property: true,

booking: true

}

});

}

async completeCheckIn(bookingId: string, data: any) {

const checkIn = await prisma.checkIn.update({

where: { bookingId },

data: {

digitalSignature: data.digitalSignature,

locationCoordinates: data.locationCoordinates,

verificationStatus: 'completed'

},

include: {

property: true,

booking: true

}

});

return checkIn;

}

async updateBookingStatus(bookingId: string, status: string) {

return await prisma.booking.update({

where: { id: bookingId },

data: { bookingStatus: status }

});

}

async scheduleCheckOutReminder(bookingId: string) {

const booking = await prisma.booking.findUnique({

where: { id: bookingId }

});

if (booking) {

const reminderTime = new Date(booking.checkOutDate);

reminderTime.setHours(9, 0, 0, 0); // 9 AM del día de check-out

await queueService.scheduleNotification({

userId: booking.userId,

type: 'check\_out\_reminder',

scheduledFor: reminderTime,

data: { bookingId }

});

}

}

async notifyPropertyOwner(propertyId: string, notificationData: any) {

const property = await prisma.property.findUnique({

where: { id: propertyId },

include: { owner: true }

});

if (property) {

await notificationService.sendToPropertyOwner(

property.owner.id,

notificationData

);

}

}

}

export const checkinService = new CheckinService();

**Panel Web Administrativo (React.js)**

**Estructura del Proyecto Frontend Web**

admin-panel/

├── src/

│ ├── components/ # Componentes reutilizables

│ │ ├── ui/ # Componentes básicos

│ │ ├── charts/ # Gráficos y visualizaciones

│ │ ├── tables/ # Tablas de datos

│ │ └── layout/ # Layout y navegación

│ ├── pages/ # Páginas principales

│ │ ├── Dashboard/ # Dashboard principal

│ │ ├── Bookings/ # Gestión de reservas

│ │ ├── Properties/ # Gestión de propiedades

│ │ ├── Reviews/ # Gestión de reseñas

│ │ └── Analytics/ # Reportes y análisis

│ ├── hooks/ # Custom hooks

│ ├── services/ # Servicios API

│ ├── store/ # Redux store

│ ├── utils/ # Utilidades

│ ├── types/ # TypeScript types

│ └── styles/ # Estilos globales

├── public/

└── package.json

**Dashboard Component**

// src/pages/Dashboard/Dashboard.tsx

import React, { useEffect, useState } from 'react';

import { Grid, Card, CardContent, Typography, Box } from '@mui/material';

import {

CheckCircle,

ExitToApp,

Hotel,

Star,

TrendingUp,

Warning

} from '@mui/icons-material';

import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, ResponsiveContainer } from 'recharts';

import { dashboardService } from '../../services/dashboardService';

import { MetricCard } from '../../components/ui/MetricCard';

import { AlertsList } from '../../components/ui/AlertsList';

import { RecentActivity } from '../../components/ui/RecentActivity';

interface DashboardData {

todayMetrics: {

checkIns: number;

checkOuts: number;

occupancy: number;

averageRating: number;

};

checkInTrend: Array<{ time: string; count: number }>;

alerts: Array<{

id: string;

type: 'warning' | 'error' | 'info';

message: string;

timestamp: Date;

}>;

recentCheckIns: Array<{

id: string;

guestName: string;

property: string;

time: string;

status: 'completed' | 'pending' | 'verifying';

}>;

}

export const Dashboard: React.FC = () => {

const [data, setData] = useState<DashboardData | null>(null);

const [loading, setLoading] = useState(true);

useEffect(() => {

loadDashboardData();

}, []);

const loadDashboardData = async () => {

try {

const dashboardData = await dashboardService.getDashboardData();

setData(dashboardData);

} catch (error) {

console.error('Error loading dashboard data:', error);

} finally {

setLoading(false);

}

};

if (loading) {

return <div>Cargando...</div>;

}

if (!data) {

return <div>Error cargando datos</div>;

}

return (

<Box sx={{ p: 3 }}>

<Typography variant="h4" sx={{ mb: 3 }}>

Dashboard - {new Date().toLocaleDateString('es-ES', {

weekday: 'long',

year: 'numeric',

month: 'long',

day: 'numeric'

})}

</Typography>

{/\* Métricas principales \*/}

<Grid container spacing={3} sx={{ mb: 4 }}>

<Grid item xs={12} sm={6} md={3}>

<MetricCard

title="Check-ins Hoy"

value={data.todayMetrics.checkIns}

icon={<CheckCircle />}

color="success"

trend={+3}

/>

</Grid>

<Grid item xs={12} sm={6} md={3}>

<MetricCard

title="Check-outs Hoy"

value={data.todayMetrics.checkOuts}

icon={<ExitToApp />}

color="info"

trend={+1}

/>

</Grid>

<Grid item xs={12} sm={6} md={