Especificaciones Técnicas de Implementación - TurisGal

1. STACK TECNOLÓGICO DETALLADO

Frontend Móvil (React Native)

Dependencias Principales

json

```json
{
  "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

typescript

```typescript
// 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

typescript

```
// 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

json

```json
{
  "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

```
// 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

typescript

```typescript
// 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

typescript

```typescript
// 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'
      };
    }
  }
```

```typescript
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

typescript

```tsx
// 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={
```