**Analytics, Notificaciones y Funcionalidades Avanzadas - TurisGal**

**1. SISTEMA DE MÉTRICAS Y ANALYTICS DETALLADO**

**Dashboard de Analytics para Propietarios**

**Métricas de Ocupación**

// src/services/AnalyticsService.ts

export class AnalyticsService {

async getOccupancyMetrics(propertyId: string, dateRange: DateRange): Promise<OccupancyMetrics> {

const metrics = await prisma.$queryRaw`

WITH daily\_metrics AS (

SELECT

date,

COUNT(DISTINCT b.id) as bookings\_count,

SUM(CASE WHEN b.booking\_status = 'checked\_in' THEN 1 ELSE 0 END) as checked\_in\_count,

SUM(b.guests\_count) as total\_guests,

AVG(b.total\_amount) as avg\_revenue\_per\_booking,

COUNT(CASE WHEN ci.verification\_status = 'verified' THEN 1 END) as verified\_checkins

FROM generate\_series(${dateRange.start}::date, ${dateRange.end}::date, '1 day') as date

LEFT JOIN bookings b ON DATE(b.check\_in\_date) = date AND b.property\_id = ${propertyId}

LEFT JOIN check\_ins ci ON ci.booking\_id = b.id

GROUP BY date

ORDER BY date

)

SELECT

\*,

LAG(bookings\_count, 7) OVER (ORDER BY date) as bookings\_prev\_week,

LAG(total\_guests, 7) OVER (ORDER BY date) as guests\_prev\_week

FROM daily\_metrics

`;

return {

dailyMetrics: metrics,

totalRevenue: metrics.reduce((sum, day) => sum + (day.avg\_revenue\_per\_booking \* day.bookings\_count || 0), 0),

averageOccupancy: this.calculateAverageOccupancy(metrics),

growthRates: this.calculateGrowthRates(metrics),

seasonalTrends: await this.getSeasonalTrends(propertyId, dateRange)

};

}

async getCheckInAnalytics(propertyId: string, dateRange: DateRange): Promise<CheckInAnalytics> {

const checkInData = await prisma.$queryRaw`

SELECT

DATE\_PART('hour', ci.check\_in\_timestamp) as hour,

COUNT(\*) as checkin\_count,

AVG(EXTRACT(EPOCH FROM (ci.check\_in\_timestamp - b.check\_in\_date))) / 60 as avg\_delay\_minutes,

COUNT(CASE WHEN ci.verification\_status = 'verified' THEN 1 END) as successful\_verifications,

COUNT(CASE WHEN ci.verification\_status = 'failed' THEN 1 END) as failed\_verifications,

AVG(EXTRACT(EPOCH FROM (ci.updated\_at - ci.created\_at))) / 60 as avg\_process\_time\_minutes

FROM check\_ins ci

JOIN bookings b ON ci.booking\_id = b.id

WHERE b.property\_id = ${propertyId}

AND ci.check\_in\_timestamp BETWEEN ${dateRange.start} AND ${dateRange.end}

GROUP BY DATE\_PART('hour', ci.check\_in\_timestamp)

ORDER BY hour

`;

const verificationAnalytics = await this.getVerificationAnalytics(propertyId, dateRange);

const deviceAnalytics = await this.getDeviceAnalytics(propertyId, dateRange);

return {

hourlyDistribution: checkInData,

verificationSuccessRate: verificationAnalytics.successRate,

averageProcessTime: checkInData.reduce((sum, h) => sum + h.avg\_process\_time\_minutes, 0) / checkInData.length,

deviceBreakdown: deviceAnalytics,

commonIssues: await this.getCommonCheckInIssues(propertyId, dateRange)

};

}

async getGuestSatisfactionMetrics(propertyId: string, dateRange: DateRange): Promise<SatisfactionMetrics> {

const reviewsData = await prisma.review.groupBy({

by: ['overall\_rating'],

where: {

propertyId,

createdAt: {

gte: dateRange.start,

lte: dateRange.end

}

},

\_count: true,

\_avg: {

cleanliness\_rating: true,

location\_rating: true,

value\_rating: true,

service\_rating: true

}

});

const npsScore = await this.calculateNPS(propertyId, dateRange);

const sentimentAnalysis = await this.analyzeSentiment(propertyId, dateRange);

return {

averageRating: await this.getAverageRating(propertyId, dateRange),

ratingDistribution: reviewsData,

npsScore,

sentimentAnalysis,

improvementAreas: await this.identifyImprovementAreas(propertyId, dateRange),

competitorComparison: await this.getCompetitorComparison(propertyId)

};

}

async getRevenueAnalytics(propertyId: string, dateRange: DateRange): Promise<RevenueAnalytics> {

const revenueData = await prisma.$queryRaw`

WITH revenue\_metrics AS (

SELECT

DATE\_TRUNC('day', b.check\_in\_date) as date,

SUM(b.total\_amount) as daily\_revenue,

COUNT(\*) as bookings\_count,

AVG(b.total\_amount) as adr, -- Average Daily Rate

SUM(b.total\_amount) / COUNT(DISTINCT r.id) as revpar -- Revenue Per Available Room

FROM bookings b

JOIN rooms r ON r.property\_id = b.property\_id

WHERE b.property\_id = ${propertyId}

AND b.check\_in\_date BETWEEN ${dateRange.start} AND ${dateRange.end}

AND b.booking\_status IN ('confirmed', 'checked\_in', 'checked\_out')

GROUP BY DATE\_TRUNC('day', b.check\_in\_date)

ORDER BY date

)

SELECT

\*,

SUM(daily\_revenue) OVER (ORDER BY date ROWS UNBOUNDED PRECEDING) as cumulative\_revenue,

LAG(daily\_revenue, 1) OVER (ORDER BY date) as prev\_day\_revenue,

LAG(daily\_revenue, 7) OVER (ORDER BY date) as prev\_week\_revenue

FROM revenue\_metrics

`;

return {

totalRevenue: revenueData.reduce((sum, day) => sum + day.daily\_revenue, 0),

averageDailyRate: revenueData.reduce((sum, day) => sum + day.adr, 0) / revenueData.length,

revenuePerAvailableRoom: revenueData.reduce((sum, day) => sum + day.revpar, 0) / revenueData.length,

dailyTrends: revenueData,

forecasting: await this.generateRevenueForecast(propertyId, revenueData),

seasonalAnalysis: await this.getSeasonalRevenueAnalysis(propertyId),

channelPerformance: await this.getChannelRevenueBreakdown(propertyId, dateRange)

};

}

private async analyzeSentiment(propertyId: string, dateRange: DateRange): Promise<SentimentAnalysis> {

const reviews = await prisma.review.findMany({

where: {

propertyId,

createdAt: { gte: dateRange.start, lte: dateRange.end },

comment: { not: null }

},

select: { comment: true, overall\_rating: true, createdAt: true }

});

// Análisis de sentimiento usando servicio externo o librería local

const sentimentResults = await Promise.all(

reviews.map(async review => {

const sentiment = await this.analyzeSentimentText(review.comment);

return {

...sentiment,

rating: review.overall\_rating,

date: review.createdAt

};

})

);

const positiveCount = sentimentResults.filter(s => s.sentiment === 'positive').length;

const negativeCount = sentimentResults.filter(s => s.sentiment === 'negative').length;

const neutralCount = sentimentResults.filter(s => s.sentiment === 'neutral').length;

return {

overallSentiment: positiveCount > negativeCount ? 'positive' : negativeCount > positiveCount ? 'negative' : 'neutral',

distribution: {

positive: positiveCount,

negative: negativeCount,

neutral: neutralCount

},

keywords: await this.extractKeywords(reviews.map(r => r.comment).join(' ')),

trends: this.analyzeSentimentTrends(sentimentResults)

};

}

async generatePredictiveInsights(propertyId: string): Promise<PredictiveInsights> {

// Machine Learning predictions usando datos históricos

const historicalData = await this.getHistoricalData(propertyId, 365); // 1 año de datos

return {

occupancyForecast: await this.predictOccupancy(historicalData),

revenueForecast: await this.predictRevenue(historicalData),

demandPeaks: await this.predictDemandPeaks(historicalData),

recommendations: await this.generateRecommendations(propertyId, historicalData)

};

}

}

// Dashboard Component for Analytics

// src/components/analytics/AnalyticsDashboard.tsx

export const AnalyticsDashboard: React.FC<{ propertyId: string }> = ({ propertyId }) => {

const [dateRange, setDateRange] = useState({

start: subDays(new Date(), 30),

end: new Date()

});

const [metrics, setMetrics] = useState(null);

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

useEffect(() => {

loadAnalytics();

}, [propertyId, dateRange]);

const loadAnalytics = async () => {

setLoading(true);

try {

const [occupancy, checkIn, satisfaction, revenue] = await Promise.all([

analyticsService.getOccupancyMetrics(propertyId, dateRange),

analyticsService.getCheckInAnalytics(propertyId, dateRange),

analyticsService.getGuestSatisfactionMetrics(propertyId, dateRange),

analyticsService.getRevenueAnalytics(propertyId, dateRange)

]);

setMetrics({ occupancy, checkIn, satisfaction, revenue });

} catch (error) {

console.error('Error loading analytics:', error);

} finally {

setLoading(false);

}

};

if (loading) return <AnalyticsLoadingSkeleton />;

return (

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

<Grid container spacing={3}>

{/\* Header con filtros de fecha \*/}

<Grid item xs={12}>

<Card>

<CardContent>

<Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>

<Typography variant="h4">Analytics Dashboard</Typography>

<DateRangePicker

value={dateRange}

onChange={setDateRange}

/>

</Box>

</CardContent>

</Card>

</Grid>

{/\* KPIs Principales \*/}

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

<MetricCard

title="Ocupación Promedio"

value={`${metrics.occupancy.averageOccupancy.toFixed(1)}%`}

trend={metrics.occupancy.growthRates.occupancy}

icon={<Hotel />}

color="primary"

/>

</Grid>

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

<MetricCard

title="Revenue Total"

value={`€${metrics.revenue.totalRevenue.toLocaleString()}`}

trend={metrics.revenue.growthRates?.revenue}

icon={<Euro />}

color="success"

/>

</Grid>

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

<MetricCard

title="Check-ins Exitosos"

value={`${metrics.checkIn.verificationSuccessRate.toFixed(1)}%`}

trend={metrics.checkIn.growthRates?.successRate}

icon={<CheckCircle />}

color="info"

/>

</Grid>

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

<MetricCard

title="Satisfacción"

value={`${metrics.satisfaction.averageRating.toFixed(1)}/5`}

trend={metrics.satisfaction.trendData?.growth}

icon={<Star />}

color="warning"

/>

</Grid>

{/\* Gráficos de Ocupación \*/}

<Grid item xs={12} md={8}>

<Card>

<CardContent>

<Typography variant="h6" sx={{ mb: 2 }}>Tendencia de Ocupación</Typography>

<ResponsiveContainer width="100%" height={400}>

<LineChart data={metrics.occupancy.dailyMetrics}>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="date" />

<YAxis />

<Tooltip />

<Line type="monotone" dataKey="bookings\_count" stroke="#1976d2" name="Reservas" />

<Line type="monotone" dataKey="checked\_in\_count" stroke="#2e7d32" name="Check-ins" />

</LineChart>

</ResponsiveContainer>

</CardContent>

</Card>

</Grid>

{/\* Distribución de Check-ins por hora \*/}

<Grid item xs={12} md={4}>

<Card>

<CardContent>

<Typography variant="h6" sx={{ mb: 2 }}>Check-ins por Hora</Typography>

<ResponsiveContainer width="100%" height={400}>

<BarChart data={metrics.checkIn.hourlyDistribution}>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="hour" />

<YAxis />

<Tooltip />

<Bar dataKey="checkin\_count" fill="#1976d2" />

</BarChart>

</ResponsiveContainer>

</CardContent>

</Card>

</Grid>

{/\* Análisis de Satisfacción \*/}

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

<Card>

<CardContent>

<Typography variant="h6" sx={{ mb: 2 }}>Distribución de Valoraciones</Typography>

<ResponsiveContainer width="100%" height={300}>

<PieChart>

<Pie

data={metrics.satisfaction.ratingDistribution}

dataKey="\_count"

nameKey="overall\_rating"

cx="50%"

cy="50%"

outerRadius={80}

fill="#8884d8"

label

/>

<Tooltip />

</PieChart>

</ResponsiveContainer>

</CardContent>

</Card>

</Grid>

{/\* Análisis de Sentimiento \*/}

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

<Card>

<CardContent>

<Typography variant="h6" sx={{ mb: 2 }}>Análisis de Sentimiento</Typography>

<Box sx={{ mb: 2 }}>

<Typography variant="body2" color="text.secondary">

Sentimiento General: {metrics.satisfaction.sentimentAnalysis.overallSentiment}

</Typography>

</Box>

<Grid container spacing={2}>

<Grid item xs={4}>

<Box sx={{ textAlign: 'center' }}>

<Typography variant="h4" color="success.main">

{metrics.satisfaction.sentimentAnalysis.distribution.positive}

</Typography>

<Typography variant="body2">Positivo</Typography>

</Box>

</Grid>

<Grid item xs={4}>

<Box sx={{ textAlign: 'center' }}>

<Typography variant="h4" color="text.secondary">

{metrics.satisfaction.sentimentAnalysis.distribution.neutral}

</Typography>

<Typography variant="body2">Neutral</Typography>

</Box>

</Grid>

<Grid item xs={4}>

<Box sx={{ textAlign: 'center' }}>

<Typography variant="h4" color="error.main">

{metrics.satisfaction.sentimentAnalysis.distribution.negative}

</Typography>

<Typography variant="body2">Negativo</Typography>

</Box>

</Grid>

</Grid>

<Box sx={{ mt: 2 }}>

<Typography variant="body2" fontWeight="bold">Palabras clave:</Typography>

<Box sx={{ display: 'flex', gap: 1, flexWrap: 'wrap', mt: 1 }}>

{metrics.satisfaction.sentimentAnalysis.keywords.slice(0, 8).map((keyword, index) => (

<Chip key={index} label={keyword} size="small" />

))}

</Box>

</Box>

</CardContent>

</Card>

</Grid>

{/\* Revenue Trends \*/}

<Grid item xs={12}>

<Card>

<CardContent>

<Typography variant="h6" sx={{ mb: 2 }}>Tendencias de Ingresos</Typography>

<ResponsiveContainer width="100%" height={400}>

<ComposedChart data={metrics.revenue.dailyTrends}>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="date" />

<YAxis yAxisId="left" />

<YAxis yAxisId="right" orientation="right" />

<Tooltip />

<Bar yAxisId="left" dataKey="daily\_revenue" fill="#1976d2" name="Ingresos Diarios (€)" />

<Line yAxisId="right" type="monotone" dataKey="adr" stroke="#ff7300" name="ADR (€)" />

<Line yAxisId="right" type="monotone" dataKey="revpar" stroke="#8884d8" name="RevPAR (€)" />

</ComposedChart>

</ResponsiveContainer>

</CardContent>

</Card>

</Grid>

{/\* Insights y Recomendaciones \*/}

<Grid item xs={12}>

<Card>

<CardContent>

<Typography variant="h6" sx={{ mb: 2 }}>Insights y Recomendaciones</Typography>

<Grid container spacing={2}>

<Grid item xs={12} md={4}>

<Box sx={{ p: 2, bgcolor: 'success.light', borderRadius: 1 }}>

<Typography variant="subtitle1" sx={{ mb: 1 }}>💡 Oportunidad</Typography>

<Typography variant="body2">

Los check-ins entre 16:00-18:00 tienen mayor tasa de éxito.

Considera incentivar check-ins en este horario.

</Typography>

</Box>

</Grid>

<Grid item xs={12} md={4}>

<Box sx={{ p: 2, bgcolor: 'warning.light', borderRadius: 1 }}>

<Typography variant="subtitle1" sx={{ mb: 1 }}>⚠️ Atención</Typography>

<Typography variant="body2">

La valoración de limpieza ha bajado 0.3 puntos este mes.

Revisar protocolos de limpieza.

</Typography>

</Box>

</Grid>

<Grid item xs={12} md={4}>

<Box sx={{ p: 2, bgcolor: 'info.light', borderRadius: 1 }}>

<Typography variant="subtitle1" sx={{ mb: 1 }}>📈 Tendencia</Typography>

<Typography variant="body2">

Los huéspedes valoran especialmente la ubicación.

Destaca este aspecto en tu marketing.

</Typography>

</Box>

</Grid>

</Grid>

</CardContent>

</Card>

</Grid>

</Grid>

</Box>

);

};

**2. SISTEMA DE NOTIFICACIONES AVANZADO**

**Engine de Notificaciones Inteligentes**

// src/services/NotificationService.ts

export class NotificationService {

private templates: Map<string, NotificationTemplate> = new Map();

private channels: Map<string, NotificationChannel> = new Map();

private queue = new Queue('notifications');

constructor() {

this.initializeTemplates();

this.initializeChannels();

this.setupNotificationQueue();

}

private initializeTemplates(): void {

// Plantillas para diferentes tipos de notificaciones

this.templates.set('check\_in\_reminder', {

title: {

es: 'Recordatorio de Check-in',

en: 'Check-in Reminder',

gl: 'Lembrete de Check-in'

},

body: {

es: 'Tu check-in es mañana en {{propertyName}}. ¿Todo listo?',

en: 'Your check-in is tomorrow at {{propertyName}}. All set?',

gl: 'O teu check-in é mañá en {{propertyName}}. Todo preparado?'

},

actions: [

{ id: 'view\_booking', label: 'Ver reserva', deepLink: '/booking/{{bookingId}}' },

{ id: 'contact\_property', label: 'Contactar hotel', action: 'call:{{propertyPhone}}' }

],

priority: 'normal',

channels: ['push', 'email'],

timing: {

send\_at: 'checkin\_date\_minus\_1\_day\_at\_18:00'

}

});

this.templates.set('check\_out\_reminder', {

title: {

es: 'Hora del Check-out',

en: 'Check-out Time',

gl: 'Hora do Check-out'

},

body: {

es: 'Recuerda hacer el check-out antes de las {{checkoutTime}}',

en: 'Remember to check-out before {{checkoutTime}}',

gl: 'Lembra facer o check-out antes das {{checkoutTime}}'

},

actions: [

{ id: 'start\_checkout', label: 'Iniciar check-out', deepLink: '/checkout/{{bookingId}}' }

],

priority: 'high',

channels: ['push'],

timing: {

send\_at: 'checkout\_date\_at\_09:00'

}

});

this.templates.set('review\_request', {

title: {

es: '¿Cómo fue tu estancia?',

en: 'How was your stay?',

gl: 'Como foi a túa estadía?'

},

body: {

es: 'Ayuda a otros viajeros compartiendo tu experiencia en {{propertyName}}',

en: 'Help other travelers by sharing your experience at {{propertyName}}',

gl: 'Axuda a outros viaxeiros compartindo a túa experiencia en {{propertyName}}'

},

actions: [

{ id: 'write\_review', label: 'Escribir reseña', deepLink: '/review/{{bookingId}}' },

{ id: 'remind\_later', label: 'Recordar más tarde', action: 'snooze:3\_days' }

],

priority: 'low',

channels: ['push', 'email'],

timing: {

send\_at: 'checkout\_date\_plus\_1\_day\_at\_10:00'

}

});

this.templates.set('booking\_confirmed', {

title: {

es: 'Reserva confirmada',

en: 'Booking confirmed',

gl: 'Reserva confirmada'

},

body: {

es: 'Tu reserva en {{propertyName}} está confirmada para {{checkinDate}}',

en: 'Your booking at {{propertyName}} is confirmed for {{checkinDate}}',

gl: 'A túa reserva en {{propertyName}} está confirmada para {{checkinDate}}'

},

actions: [

{ id: 'view\_booking', label: 'Ver detalles', deepLink: '/booking/{{bookingId}}' },

{ id: 'add\_calendar', label: 'Añadir al calendario', action: 'calendar:{{bookingId}}' }

],

priority: 'high',

channels: ['push', 'email', 'sms'],

timing: {

send\_at: 'immediate'

}

});

this.templates.set('verification\_required', {

title: {

es: 'Verificación requerida',

en: 'Verification required',

gl: 'Verificación requirida'

},

body: {

es: 'Tu documento necesita verificación manual. Proceso en 24h.',

en: 'Your document needs manual verification. Process within 24h.',

gl: 'O teu documento precisa verificación manual. Proceso en 24h.'

},

priority: 'high',

channels: ['push', 'email'],

timing: {

send\_at: 'immediate'

}

});

// Notificaciones para propietarios

this.templates.set('new\_checkin', {

title: {

es: 'Nuevo check-in completado',

en: 'New check-in completed',

gl: 'Novo check-in completado'

},

body: {

es: '{{guestName}} ha completado el check-in en {{propertyName}}',

en: '{{guestName}} has completed check-in at {{propertyName}}',

gl: '{{guestName}} completou o check-in en {{propertyName}}'

},

channels: ['push', 'email'],

audience: 'property\_owners'

});

}

async sendNotification(

type: string,

userId: string | null,

propertyOwnerId: string | null,

data: Record<string, any>,

options?: NotificationOptions

): Promise<void> {

const template = this.templates.get(type);

if (!template) {

throw new Error(`Notification template '${type}' not found`);

}

// Determinar el público objetivo

const recipients = await this.getRecipients(userId, propertyOwnerId, template.audience);

for (const recipient of recipients) {

// Personalizar notificación para cada destinatario

const personalizedNotification = await this.personalizeNotification(template, recipient, data);

// Verificar preferencias del usuario

const userPreferences = await this.getUserNotificationPreferences(recipient.id);

const enabledChannels = template.channels.filter(channel =>

userPreferences.channels.includes(channel)

);

// Programar o enviar inmediatamente

if (template.timing?.send\_at === 'immediate' || options?.immediate) {

await this.sendImmediateNotification(personalizedNotification, enabledChannels, recipient);

} else {

await this.scheduleNotification(personalizedNotification, enabledChannels, recipient, template.timing);

}

}

}

private async personalizeNotification(

template: NotificationTemplate,

recipient: User,

data: Record<string, any>

): Promise<PersonalizedNotification> {

const language = recipient.preferredLanguage || 'es';

// Reemplazar variables en el template

const title = this.interpolateTemplate(template.title[language], data);

const body = this.interpolateTemplate(template.body[language], data);

// Personalización basada en comportamiento del usuario

const personalization = await this.getPersonalizationData(recipient.id);

return {

title,

body,

actions: template.actions?.map(action => ({

...action,

deepLink: this.interpolateTemplate(action.deepLink, data)

})),

priority: this.calculatePriority(template.priority, personalization),

personalization

};

}

private async getPersonalizationData(userId: string): Promise<PersonalizationData> {

// Analizar comportamiento del usuario para personalización

const userBehavior = await prisma.$queryRaw`

SELECT

COUNT(\*) as total\_bookings,

AVG(r.overall\_rating) as avg\_rating\_given,

MAX(b.created\_at) as last\_booking\_date,

COUNT(CASE WHEN ci.verification\_status = 'verified' THEN 1 END) as successful\_checkins,

EXTRACT(HOUR FROM AVG(ci.check\_in\_timestamp)) as preferred\_checkin\_hour

FROM bookings b

LEFT JOIN reviews r ON r.booking\_id = b.id

LEFT JOIN check\_ins ci ON ci.booking\_id = b.id

WHERE b.user\_id = ${userId}

`;

return {

userSegment: this.classifyUserSegment(userBehavior[0]),

preferredCheckInTime: userBehavior[0].preferred\_checkin\_hour,

loyaltyLevel: this.calculateLoyaltyLevel(userBehavior[0].total\_bookings),

communicationStyle: await this.inferCommunicationStyle(userId)

};

}

async scheduleNotification(

notification: PersonalizedNotification,

channels: string[],

recipient: User,

timing: NotificationTiming

): Promise<void> {

const sendDate = this.calculateSendDate(timing);

await this.queue.add('send-notification', {

notification,

channels,

recipient,

type: timing.send\_at

}, {

delay: sendDate.getTime() - Date.now(),

attempts: 3,

backoff: {

type: 'exponential',

delay: 60000 // 1 minuto

}

});

}

private async sendPushNotification(notification: PersonalizedNotification, recipient: User): Promise<void> {

const deviceTokens = await prisma.deviceToken.findMany({

where: {

userId: recipient.id,

isActive: true

}

});

for (const deviceToken of deviceTokens) {

try {

await admin.messaging().send({

token: deviceToken.token,

notification: {

title: notification.title,

body: notification.body

},

data: {

type: notification.type,

deepLink: notification.actions?.[0]?.deepLink || '',

bookingId: notification.data?.bookingId || ''

},

android: {

priority: 'high',

notification: {

icon: 'ic\_notification',

color: '#1976d2',

sound: 'default',

clickAction: notification.actions?.[0]?.deepLink

}

},

apns: {

payload: {

aps: {

alert: {

title: notification.title,

body: notification.body

},

badge: await this.getUnreadNotificationCount(recipient.id),

sound: 'default',

category: notification.type

}

}

}

});

// Registrar entrega exitosa

await this.logNotificationDelivery(recipient.id, 'push', 'delivered');

} catch (error) {

console.error(`Failed to send push notification to ${deviceToken.token}:`, error);

// Marcar token como inactivo si es un error de token inválido

if (error.code === 'messaging/invalid-registration-token') {

await prisma.deviceToken.update({

where: { id: deviceToken.id },

data: { isActive: false }

});

}

await this.logNotificationDelivery(recipient.id, 'push', 'failed', error.message);

}

}

}

private async sendEmailNotification(notification: PersonalizedNotification, recipient: User): Promise<void> {

const emailContent = await this.generateEmailContent(notification, recipient);

try {

await this.emailService.send({

to: recipient.email,

subject: notification.title,

html: emailContent.html,

text: emailContent.text,

templateData: {

userName: recipient.firstName,

notification: notification,

unsubscribeUrl: `${process.env.BASE\_URL}/unsubscribe/${recipient.id}`

}

});

await this.logNotificationDelivery(recipient.id, 'email', 'delivered');

} catch (error) {

console.error(`Failed to send email to ${recipient.email}:`, error);

await this.logNotificationDelivery(recipient.id, 'email', 'failed', error.message);

}

}

private async sendSMSNotification(notification: PersonalizedNotification, recipient: User): Promise<void> {

if (!recipient.phone) return;

try {

await this.smsService.send({

to: recipient.phone,

body: `${notification.title}\n\n${notification.body}\n\nTurisGal`

});

await this.logNotificationDelivery(recipient.id, 'sms', 'delivered');

} catch (error) {

console.error(`Failed to send SMS to ${recipient.phone}:`, error);

await this.logNotificationDelivery(recipient.id, 'sms', 'failed', error.message);

}

}

// Notificaciones inteligentes basadas en contexto

async sendContextualNotification(context: NotificationContext): Promise<void> {

const { userId, location, timeOfDay, weather, bookingId } = context;

// Análisis de contexto para determinar la mejor notificación

if (location && bookingId) {

const booking = await prisma.booking.findUnique({

where: { id: bookingId },

include: { property: true }

});

if (booking) {

const distanceToProperty = this.calculateDistance(location, booking.property.coordinates);

// Notificación de proximidad

if (distanceToProperty < 1000 && timeOfDay > 14) { // Menos de 1km y después de las 2 PM

await this.sendNotification('proximity\_checkin', userId, null, {

propertyName: booking.property.name,

distance: Math.round(distanceToProperty),

bookingId: booking.id

});

}

// Notificación de clima

if (weather?.condition === 'rain' && booking.property.amenities.includes('umbrella')) {

await this.sendNotification('weather\_amenity', userId, null, {

propertyName: booking.property.name,

amenity: 'paraguas disponibles en recepción',

weather: weather.condition

});

}

}

}

}

// Analytics de notificaciones

async getNotificationAnalytics(propertyId?: string, dateRange?: DateRange): Promise<NotificationAnalytics> {

const analytics = await prisma.$queryRaw`

SELECT

notification\_type,

channel,

status,

COUNT(\*) as count,

AVG(CASE WHEN clicked\_at IS NOT NULL THEN 1 ELSE 0 END) as click\_rate,

AVG(EXTRACT(EPOCH FROM (clicked\_at - sent\_at))) as avg\_click\_time

FROM notification\_logs nl

JOIN notifications n ON n.id = nl.notification\_id

WHERE (${propertyId} IS NULL OR n.property\_id = ${propertyId})

AND (${dateRange?.start} IS NULL OR nl.sent\_at >= ${dateRange?.start})

AND (${dateRange?.end} IS NULL OR nl.sent\_at <= ${dateRange?.end})

GROUP BY notification\_type, channel, status

ORDER BY count DESC

`;

return {

deliveryRates: this.calculateDeliveryRates(analytics),

engagementRates: this.calculateEngagementRates(analytics),

channelPerformance: this.analyzeChannelPerformance(analytics),

typePerformance: this.analyzeTypePerformance(analytics),

recommendations: this.generateNotificationRecommendations(analytics)

};

}

}

// Configuración de preferencias de usuario

// src/components/NotificationPreferences.tsx

export const NotificationPreferences: React.FC<{ userId: string }> = ({ userId }) => {

const [preferences, setPreferences] = useState<NotificationPreferences | null>(null);

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

useEffect(() => {

loadPreferences();

}, [userId]);

const loadPreferences = async () => {

try {

const userPrefs = await notificationService.getUserPreferences(userId);

setPreferences(userPrefs);

} catch (error) {

console.error('Error loading preferences:', error);

} finally {

setLoading(false);

}

};

const updatePreference = async (category: string, channel: string, enabled: boolean) => {

try {

await notificationService.updateUserPreferences(userId, {

[category]: {

...preferences[category],

[channel]: enabled

}

});

setPreferences(prev => ({

...prev,

[category]: {

...prev[category],

[channel]: enabled

}

}));

} catch (error) {

console.error('Error updating preferences:', error);

}

};

if (loading) return <CircularProgress />;

return (

<Card>

<CardContent>

<Typography variant="h6" sx={{ mb: 3 }}>Preferencias de Notificaciones</Typography>

{Object.entries(notificationCategories).map(([category, config]) => (

<Box key={category} sx={{ mb: 3 }}>

<Typography variant="subtitle1" sx={{ mb: 1 }}>

{config.icon} {config.title}

</Typography>

<Typography variant="body2" color="text.secondary" sx={{ mb: 2 }}>

{config.description}

</Typography>

<Grid container spacing={2}>

{config.channels.map(channel => (

<Grid item xs={12} sm={4} key={channel}>

<FormControlLabel

control={

<Switch

checked={preferences[category]?.[channel] || false}

onChange={(e) => updatePreference(category, channel, e.target.checked)}

/>

}

label={channelLabels[channel]}

/>

</Grid>

))}

</Grid>

<Divider sx={{ mt: 2 }} />

</Box>

))}

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

<Typography variant="subtitle1" sx={{ mb: 2 }}>Horarios de Silencio</Typography>

<Grid container spacing={2}>

<Grid item xs={6}>

<TimePicker

label="No molestar desde"

value={preferences.quietHours?.start}

onChange={(time) => updatePreference('quietHours', 'start', time)}

/>

</Grid>

<Grid item xs={6}>

<TimePicker

label="No molestar hasta"

value={preferences.quietHours?.end}

onChange={(time) => updatePreference('quietHours', 'end', time)}

/>

</Grid>

</Grid>

</Box>

</CardContent>

</Card>

);

};

const notificationCategories = {

bookings: {

title: 'Reservas',

description: 'Confirmaciones, modificaciones y cancelaciones',

icon: '📋',

channels: ['push', 'email', 'sms']

},

checkin: {

title: 'Check-in',

description: 'Recordatorios y actualizaciones del proceso',

icon: '📱',

channels: ['push', 'email']

},

reviews: {

title: 'Reseñas',

description: 'Solicitudes para escribir reseñas',

icon: '⭐',

channels: ['push', 'email']

},

marketing: {

title: 'Ofertas y promociones',

description: 'Descuentos y ofertas especiales',

icon: '🎁',

channels: ['push', 'email']

}

};

Continúo con más funcionalidades avanzadas. ¿Quieres que siga con el sistema de pagos, gestión de usuarios y roles, o algún otro aspecto específico?