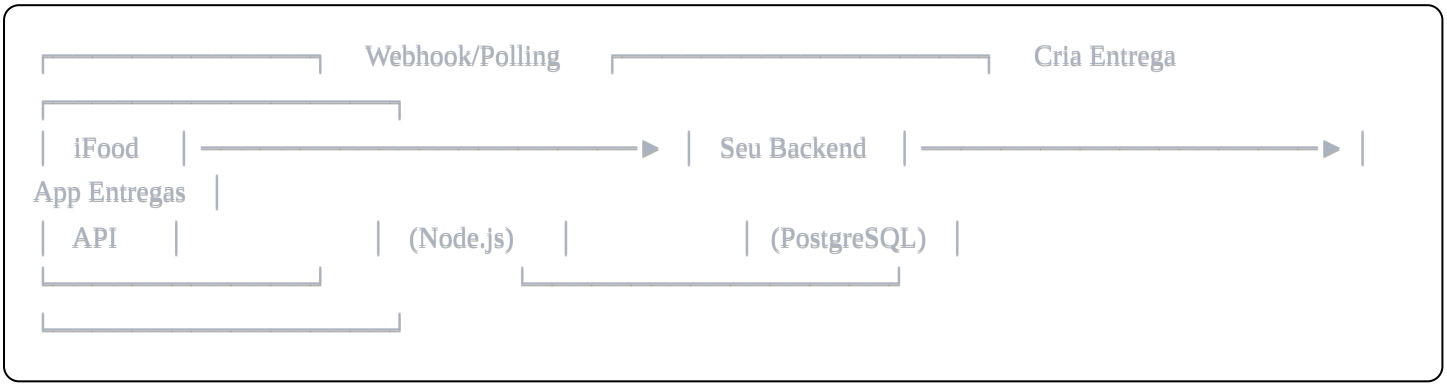
Integração iFood → App de Entregas

Visão Geral

Quando um pedido no iFood mudar para status **READY_TO_PICKUP** (pronto) ou **DISPATCHED** (saiu para entrega), uma nova entrega será criada automaticamente no seu app.

Arquitetura da Solução



Fluxo de Status dos Pedidos iFood

Status	Código	Descrição	Ação no seu App
PLACED	PLC	Pedido criado	-
CONFIRMED	CFM	Pedido confirmado	-
READY_TO_PICKUP	RTP	Pedido pronto para coleta	**Criar entrega**
DISPATCHED	DSP	Pedido saiu para entrega	**Criar entrega** (alternativa)
CONCLUDED	CON	Pedido concluído	-
CANCELLED	CAN	Pedido cancelado	Cancelar entrega se existir

---

1. Configuração Inicial no iFood

1.1 Cadastro no Portal Developer

1. Acesse: https://developer.ifood.com.br
2. Crie conta com **Perfil Profissional** (CNPJ obrigatório)
3. O portal cria automaticamente um **app de teste** com credenciais

1.2 Credenciais Necessárias

env

```
IFOOD_CLIENT_ID=seu_client_id
IFOOD_CLIENT_SECRET=seu_client_secret
IFOOD_MERCHANT_ID=id_da_loja
```

---

2. Autenticação

2.1 Obter Access Token

javascript

```
// src/services/ifood/auth.js
const axios = require('axios');

const IFOOD_BASE_URL = 'https://merchant-api.ifood.com.br';

let tokenCache = {
  accessToken: null,
  expiresAt: null
};

async function getAccessToken() {
  // Verifica se token ainda é válido
  if (tokenCache.accessToken && tokenCache.expiresAt > Date.now()) {
    return tokenCache.accessToken;
  }

  const response = await axios.post(
    `${IFOOD_BASE_URL}/authentication/v1.0/oauth/token`,
    new URLSearchParams({
      grantType: 'client_credentials',
      clientId: process.env.IFOOD_CLIENT_ID,
      clientSecret: process.env.IFOOD_CLIENT_SECRET
    }),
    {
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      }
    }
  );

  // Guarda token em cache
  tokenCache = {
    accessToken: response.data.accessToken,
    expiresAt: Date.now() + (response.data.expiresIn * 1000) - 60000 // 1min antes
  };

  return tokenCache.accessToken;
}

module.exports = { getAccessToken };
```

---

3. Recebendo Eventos (2 Opções)

3.1 Opção A: Webhook (Recomendado)

O iFood envia eventos automaticamente para sua URL.

Configuração no Portal Developer:

1. Vá em Meus Apps → Seu App → Webhook
2. Configure URL: `https://seu-dominio.com/webhooks/ifood`
3. Ative os eventos desejados

Endpoint no seu backend:

```javascript
```

```
// src/routes/webhooks/ifood.js
const express = require('express');
const crypto = require('crypto');
const router = express.Router();
const { processOrderEvent } = require('../../services/ifood/orderProcessor');

// Middleware para validar assinatura do iFood
function validateIFoodSignature(req, res, next) {
  const signature = req.headers['x-ifood-signature'];
  const body = JSON.stringify(req.body);

  const expectedSignature = crypto
    .createHmac('sha256', process.env.IFOOD_CLIENT_SECRET)
    .update(body)
    .digest('hex');

  if (signature !== expectedSignature) {
    return res.status(401).json({ error: 'Invalid signature' });
  }

  next();
}

router.post('/webhooks/ifood',
  express.json({ verify: (req, res, buf) => { req.rawBody = buf; } }),
  validateIFoodSignature,
  async (req, res) => {
    try {
      const event = req.body;

      console.log(`Evento iFood recebido: ${event.fullCode}`, {
        orderId: event.orderId,
        merchantId: event.merchantId
      });

      // Processa o evento de forma assíncrona
      processOrderEvent(event).catch(err => {
        console.error('Erro ao processar evento:', err);
      });

      // Responde imediatamente com 202 (obrigatório em até 5 segundos)
      res.status(202).send();

    } catch (error) {
      console.error('Erro no webhook:', error);
      res.status(500).json({ error: error.message });
```

```
    }
  }
);

module.exports = router;
```

3.2 Opção B: Polling (Fallback)

Consulta periódica à API do iFood.

```javascript
```

```
// src/services/ifood/polling.js
const axios = require('axios');
const { getAccessToken } = require('./auth');
const { processOrderEvent } = require('./orderProcessor');

const IFOOD_BASE_URL = 'https://merchant-api.ifood.com.br';

async function pollEvents() {
  try {
    const token = await getAccessToken();

    // Busca eventos pendentes
    const response = await axios.get(
      `${IFOOD_BASE_URL}/events/v1.0/events:polling`,
      {
        headers: { Authorization: `Bearer ${token}` },
        params: {
          excludeHeartbeat: true // Importante: evita manter loja aberta indevidamente
        }
      }
    );

    const events = response.data || [];
    const eventIds = [];

    for (const event of events) {
      console.log(`Evento recebido: ${event.fullCode}`, event.orderId);

      await processOrderEvent(event);
      eventIds.push(event.id);
    }

    // Confirma recebimento dos eventos (acknowledgment)
    if (eventIds.length > 0) {
      await axios.post(
        `${IFOOD_BASE_URL}/events/v1.0/events/acknowledgment`,
        eventIds,
        {
          headers: { Authorization: `Bearer ${token}` }
        }
      );
    }

  } catch (error) {
    console.error('Erro no polling:', error.message);
  }
```

```
}

// Executa a cada 30 segundos (obrigatório pelo iFood)
function startPolling() {
  setInterval(pollEvents, 30000);
  pollEvents(); // Executa imediatamente
}

module.exports = { startPolling, pollEvents };
```

---

4. Processador de Eventos

javascript

```javascript
// src/services/ifood/orderProcessor.js
const { getOrderDetails } = require('./orderService');
const { createDelivery } = require('../delivery/deliveryService');

// Eventos que disparam criação de entrega
const TRIGGER_EVENTS = ['READY_TO_PICKUP', 'DISPATCHED'];

async function processOrderEvent(event) {
  const { fullCode, orderId, merchantId } = event;

  // Verifica se é um evento que dispara criação de entrega
  if (!TRIGGER_EVENTS.includes(fullCode)) {
    console.log(`Evento ${fullCode} ignorado - não é gatilho de entrega`);
    return;
  }

  // Verifica se já existe entrega para este pedido (evita duplicação)
  const existingDelivery = await findDeliveryByExternalId(orderId);
  if (existingDelivery) {
    console.log(`Entrega já existe para pedido ${orderId}`);
    return;
  }

  // Busca detalhes completos do pedido
  const orderDetails = await getOrderDetails(orderId);

  // Cria a entrega no seu app
  await createDeliveryFromIFoodOrder(orderDetails, merchantId);
}

async function findDeliveryByExternalId(externalId) {
  const { pool } = require('../../database');
  const result = await pool.query(
    'SELECT id FROM deliveries WHERE external_id = $1 AND external_source = $2',
    [externalId, 'ifood']
  );
  return result.rows[0];
}

module.exports = { processOrderEvent };
```

---

5. Buscar Detalhes do Pedido

javascript

```
// src/services/ifood/orderService.js
const axios = require('axios');
const { getAccessToken } = require('./auth');

const IFOOD_BASE_URL = 'https://merchant-api.ifood.com.br';

async function getOrderDetails(orderId) {
  const token = await getAccessToken();

  const response = await axios.get(
    `${IFOOD_BASE_URL}/order/v1.0/orders/${orderId}`,
    {
      headers: { Authorization: `Bearer ${token}` }
    }
  );

  return response.data;
}

module.exports = { getOrderDetails };
```

Estrutura do pedido retornado:

javascript

```
{
  "id": "63895716-37c3-4372-afd0-3240bfef708d",
  "displayId": "XPTO",
  "orderType": "DELIVERY",
  "orderTiming": "IMMEDIATE", // ou SCHEDULED
  "salesChannel": "IFOOD",
  "createdAt": "2021-02-16T18:10:27Z",

  "merchant": {
    "id": "c54bb20a-bce0-4e38-bd4a-fe5f0a7b6b5a",
    "name": "Nome do Restaurante"
  },

  "customer": {
    "id": "22587f70-60b4-423c-8cd2-27d288f47f99",
    "name": "Nome do Cliente",
    "phone": {
      "number": "0800 XXX XXXX",
      "localizer": "27534642" // Código para ligar via iFood
    }
  },

  "delivery": {
    "deliveredBy": "IFOOD", // ou MERCHANT (entrega própria)
    "deliveryDateTime": "2021-02-09T18:10:32Z",
    "observations": "Deixar na portaria",
    "deliveryAddress": {
      "streetName": "Rua Exemplo",
      "streetNumber": "1234",
      "formattedAddress": "Rua Exemplo, 1234, Apto 101",
      "neighborhood": "Centro",
      "complement": "Apto 101",
      "reference": "Perto da praça",
      "postalCode": "12345678",
      "city": "São Paulo",
      "state": "SP",
      "coordinates": {
        "latitude": -23.550520,
        "longitude": -46.633308
      }
    },
    "pickupCode": "1234" // Código de retirada
  },

  "total": {
    "subTotal": 5000, // Em centavos (R$ 50,00)
```

```
    "deliveryFee": 500,
    "benefits": 0,
    "orderAmount": 5500
  },

  "items": [
    {
      "name": "X-Burger",
      "quantity": 2,
      "unitPrice": 2500,
      "totalPrice": 5000
    }
  ]
}
```

---

6. Criar Entrega no seu App

javascript

```
// src/services/delivery/deliveryService.js
const { pool } = require('../../database');

async function createDeliveryFromIFoodOrder(order, merchantId) {
  const client = await pool.connect();

  try {
    await client.query('BEGIN');

    // Busca a empresa vinculada ao merchant iFood
    const companyResult = await client.query(
      'SELECT id, name, address, lat, lng FROM companies WHERE ifood_merchant_id = $1',
      [merchantId]
    );

    if (companyResult.rows.length === 0) {
      throw new Error(`Empresa não encontrada para merchant iFood: ${merchantId}`);
    }

    const company = companyResult.rows[0];
    const delivery = order.delivery;
    const address = delivery.deliveryAddress;

    // Calcula valor da entrega (você pode ter sua própria lógica)
    const deliveryFee = await calculateDeliveryFee(
      company.lat,
      company.lng,
      address.coordinates.latitude,
      address.coordinates.longitude
    );

    // Cria a entrega
    const result = await client.query(`
      INSERT INTO deliveries (
        company_id,
        external_id,
        external_source,
        external_display_id,
        status,

        -- Origem (restaurante)
        origin_address,
        origin_lat,
        origin_lng,

        -- Destino (cliente)
```

```
    destination_address,
    destination_complement,
    destination_reference,
    destination_neighborhood,
    destination_city,
    destination_state,
    destination_zipcode,
    destination_lat,
    destination_lng,

    -- Cliente
    customer_name,
    customer_phone,

    -- Valores
    delivery_fee,
    order_value,

    -- Observações
    observations,
    pickup_code,

    -- Datas
    scheduled_for,
    created_at
  ) VALUES (
    $1, $2, $3, $4, $5,
    $6, $7, $8,
    $9, $10, $11, $12, $13, $14, $15, $16, $17,
    $18, $19,
    $20, $21,
    $22, $23,
    $24, NOW()
  )
  RETURNING *
`, [
  company.id,
  order.id,
  'ifood',
  order.displayId,
  'pending', // Status inicial da entrega

  company.address,
  company.lat,
  company.lng,

  address.formattedAddress,
```

```
      address.complement,
      address.reference,
      address.neighborhood,
      address.city,
      address.state,
      address.postalCode,
      address.coordinates?.latitude,
      address.coordinates?.longitude,

      order.customer.name,
      order.customer.phone?.number,

      deliveryFee,
      order.total?.orderAmount,

      delivery.observations,
      delivery.pickupCode,

      order.orderTiming === 'SCHEDULED' ? delivery.deliveryDateTime : null
    ]);

    await client.query('COMMIT');

    const newDelivery = result.rows[0];

    console.log(`✅ Entrega criada: #${newDelivery.id} para pedido iFood ${order.displayId}`);

    // Notifica entregadores disponíveis
    await notifyAvailableDrivers(newDelivery);

    return newDelivery;

  } catch (error) {
    await client.query('ROLLBACK');
    console.error('Erro ao criar entrega:', error);
    throw error;
  } finally {
    client.release();
  }
}

async function calculateDeliveryFee(originLat, originLng, destLat, destLng) {
  // Implementar sua lógica de cálculo de frete
  // Pode usar distância, zona, etc.
  return 800; // R$ 8,00 em centavos
}
```

```
async function notifyAvailableDrivers(delivery) {
  // Implementar notificação push/websocket para entregadores
  console.log(`Notificando entregadores sobre entrega #${delivery.id}`);
}

module.exports = { createDeliveryFromIFoodOrder };
```

---

7. Tabela de Vínculo Empresa ↔ iFood

sql

```
-- Adicionar coluna na tabela de empresas
ALTER TABLE companies
ADD COLUMN ifood_merchant_id VARCHAR(100) UNIQUE;

-- Criar índice
CREATE INDEX idx_companies_ifood_merchant ON companies(ifood_merchant_id);
```

---

8. Tratamento de Cancelamentos

javascript

```
// Adicionar no orderProcessor.js

async function processOrderEvent(event) {
  const { fullCode, orderId } = event;

  // Criação de entrega
  if (TRIGGER_EVENTS.includes(fullCode)) {
    // ... código anterior
  }

  // Cancelamento do pedido
  if (fullCode === 'CANCELLED') {
    await cancelDeliveryByExternalId(orderId);
  }
}

async function cancelDeliveryByExternalId(externalId) {
  const { pool } = require('../../database');

  const result = await pool.query(`
    UPDATE deliveries
    SET status = 'cancelled',
        cancelled_at = NOW(),
        cancellation_reason = 'Pedido cancelado no iFood'
    WHERE external_id = $1
      AND external_source = 'ifood'
      AND status NOT IN ('delivered', 'cancelled')
    RETURNING id
  `, [externalId]);

  if (result.rows.length > 0) {
    console.log(`❌ Entrega #${result.rows[0].id} cancelada (pedido iFood cancelado)`);

    // Se já tinha entregador, notificar
    // await notifyDriverAboutCancellation(result.rows[0].id);
  }
}
```

---

9. Configuração Completa

9.1 Arquivo .env

```
env
```

```
# iFood
IFOOD_CLIENT_ID=seu_client_id
IFOOD_CLIENT_SECRET=seu_client_secret
IFOOD_BASE_URL=https://merchant-api.ifood.com.br

# Banco de dados
DATABASE_URL=postgresql://user:pass@localhost:5432/deliveries

# Webhook
WEBHOOK_SECRET=seu_secret_para_validacao
```

9.2 Estrutura de Pastas

```
src/
├── routes/
│   └── webhooks/
│       └── ifood.js
├── services/
│   ├── ifood/
│   │   ├── auth.js
│   │   ├── polling.js
│   │   ├── orderService.js
│   │   └── orderProcessor.js
│   └── delivery/
│       └── deliveryService.js
└── database/
    └── index.js
```

---

10. Checklist de Homologação iFood

Para ter o app aprovado no iFood, você precisa:

- [ ] Fazer polling a cada 30 segundos
- [ ] Enviar acknowledgment para todos eventos recebidos
- [ ] Receber e processar pedidos DELIVERY imediatos
- [ ] Receber e processar pedidos DELIVERY agendados
- [ ] Tratar cancelamentos
- [ ] Respeitar rate limits da API
- [ ] Renovar token somente quando expirar

---

11. Links Úteis

- **Portal Developer:** https://developer.ifood.com.br
- **Documentação API:** https://developer.ifood.com.br/docs/guides
- **API Reference:** https://developer.ifood.com.br/reference
- **Eventos de Pedido:** https://developer.ifood.com.br/docs/guides/order/events

---

Próximos Passos

1. **Criar conta** no Portal Developer do iFood
2. **Configurar webhook** ou implementar polling
3. **Vincular empresas** ao `ifood_merchant_id`
4. **Testar** com pedidos de teste no ambiente sandbox
5. **Solicitar homologação** após testes completos