Planejamento do Backend com Next.js

Visão Geral

Este documento detalha o planejamento do backend do e-commerce utilizando Next.js, conforme solicitado. O backend será responsável por fornecer APIs RESTful para o frontend, gerenciar a lógica de negócios, interagir com o banco de dados MySQL e integrar com serviços externos como o Mercado Pago.

Estrutura do Projeto Next.js

```
/backend
 /src
  /app
   /api
    /auth
     /[...nextauth]
      route.js
    /products
     /[id]
      route.js
     /categories
      /[id]
       route.js
      route.js
     /featured
      route.js
     /new
      route.js
     /offers
      route.js
     /related
      route.js
     /reviews
      /[id]
       route.js
      route.js
     route.js
    /categories
     /[id]
      route.js
     /featured
      route.js
     route.js
```

```
/cart
  /items
    /[id]
      route.js
    route.js
  route.js
/orders
  /[id]
    /status
      route.js
    route.js
  route.js
/users
  /[id]
    /addresses
      /[addressId]
        route.js
      route.js
    /orders
      route.js
    /wishlist
      route.js
    route.js
  /me
    route.js
  route.js
/checkout
  /validate
    route.js
  /process
    route.js
  route.js
/payment
  /mercadopago
    /webhook
      route.js
    /preferences
      route.js
    route.js
/shipping
  /calculate
    route.js
  route.js
/search
  route.js
/webhooks
  /mercadopago
    route.js
/admin
  /dashboard
    page.js
  /products
```

```
        /[id]
          /edit
            page.js
          page.js
        /new
          page.js
        page.js
      /categories
        /[id]
          /edit
            page.js
          page.js
        /new
          page.js
        page.js
      /orders
        /[id]
          page.js
        page.js
      /customers
        /[id]
          page.js
        page.js
      /settings
        page.js
      layout.js
      page.js
    layout.js
    page.js
  /components
    /admin
      /layout
        Sidebar.jsx
        Header.jsx
        Footer.jsx
      /products
        ProductForm.jsx
        ProductList.jsx
        ProductGallery.jsx
      /orders
        OrderDetails.jsx
        OrderList.jsx
        OrderStatus.jsx
      /customers
        CustomerDetails.jsx
        CustomerList.jsx
      /dashboard
        SalesChart.jsx
        RecentOrders.jsx
        TopProducts.jsx
        Statistics.jsx
    /ui
```

```
      Button.jsx
      Input.jsx
      Select.jsx
      Modal.jsx
      Table.jsx
      Pagination.jsx
      Alert.jsx
      Spinner.jsx
    /lib
      /prisma
        index.js
        schema.prisma
      /auth
        auth.js
        permissions.js
      /mercadopago
        index.js
        webhook.js
      /utils
        formatter.js
        validator.js
        pagination.js
        filters.js
    /models
      Product.js
      Category.js
      User.js
      Order.js
      Cart.js
      Review.js
    /services
      ProductService.js
      CategoryService.js
      UserService.js
      OrderService.js
      CartService.js
      PaymentService.js
      ShippingService.js
      SearchService.js
    /middleware.js
  /prisma
    schema.prisma
    migrations/
  /public
    /images
    /icons
    /admin
  .env
  .env.local
  next.config.js
  package.json
```

Configuração do Prisma ORM

O Prisma será utilizado como ORM para interagir com o banco de dados MySQL:

```
// /lib/prisma/index.js
import { PrismaClient } from '@prisma/client';

let prisma;

if (process.env.NODE_ENV === 'production') {
  prisma = new PrismaClient();
} else {
  // Evitar múltiplas instâncias do Prisma Client em desenvolvimento
  if (!global.prisma) {
    global.prisma = new PrismaClient();
  }
  prisma = global.prisma;
}

export default prisma;
```

O schema do Prisma será baseado na estrutura do banco de dados MySQL definida anteriormente:

```
// /prisma/schema.prisma
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model User {
  id              Int      @id @default(autoincrement())
  name            String
  email           String   @unique
  password        String
  cpf             String?  @unique
  phone           String?
  birthDate       DateTime?
  type            UserType  @default(CUSTOMER)
  status          UserStatus @default(ACTIVE)
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt
  lastLogin       DateTime?
  resetPasswordToken String?
```

```
  resetPasswordExpiry DateTime?
  preferences      Json?

  addresses        Address[]
  orders           Order[]
  reviews          Review[]
  wishlistItems    Wishlist[]
  cartItems        Cart[]

  @@index([email])
  @@index([type])
  @@map("users")
}

enum UserType {
  CUSTOMER
  ADMIN
}

enum UserStatus {
  ACTIVE
  INACTIVE
  BLOCKED
}

model Address {
  id           Int      @id @default(autoincrement())
  userId       Int
  name         String
  zipCode      String
  street       String
  number       String
  complement   String?
  neighborhood String
  city         String
  state        String
  country      String   @default("Brasil")
  phone        String?
  type         AddressType @default(SHIPPING)
  isDefault    Boolean  @default(false)
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt

  user         User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  shippingOrders Order[] @relation("ShippingAddress")
  billingOrders Order[] @relation("BillingAddress")

  @@index([userId])
  @@map("addresses")
}

enum AddressType {
```

```
  SHIPPING
  BILLING
  BOTH
}

model Category {
  id          Int      @id @default(autoincrement())
  name        String
  description String?  @db.Text
  slug        String   @unique
  image       String?
  banner      String?
  parentId    Int?
  level       Int      @default(1)
  order       Int      @default(0)
  status      CategoryStatus @default(ACTIVE)
  metaTitle   String?
  metaDescription String?
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  parent      Category? @relation("CategoryToCategory", fields: [parentId],
references: [id], onDelete: SetNull)
  children    Category[] @relation("CategoryToCategory")
  products    ProductCategory[]
  coupons     CouponCategory[]

  @@index([slug])
  @@index([parentId])
  @@index([status])
  @@map("categories")
}

enum CategoryStatus {
  ACTIVE
  INACTIVE
}

model Product {
  id              Int      @id @default(autoincrement())
  sku             String   @unique
  name            String
  shortDescription  String?
  longDescription   String?  @db.Text
  price           Decimal   @db.Decimal(10, 2)
  promotionalPrice  Decimal?  @db.Decimal(10, 2)
  stock           Int      @default(0)
  weight          Decimal?  @db.Decimal(10, 2)
  height          Decimal?  @db.Decimal(10, 2)
  width           Decimal?  @db.Decimal(10, 2)
  depth           Decimal?  @db.Decimal(10, 2)
  isFeatured      Boolean   @default(false)
```

```
  isNew          Boolean   @default(false)
  status         ProductStatus @default(ACTIVE)
  manageStock    Boolean   @default(true)
  allowOutOfStock Boolean  @default(false)
  slug           String    @unique
  metaTitle      String?
  metaDescription String?
  createdAt      DateTime  @default(now())
  updatedAt      DateTime  @updatedAt
  views          Int       @default(0)
  sales          Int       @default(0)
  attributes     Json?

  images         ProductImage[]
  categories     ProductCategory[]
  variants       ProductVariant[]
  reviews        Review[]
  orderItems     OrderItem[]
  wishlistItems  Wishlist[]
  cartItems      Cart[]
  productAttributes ProductAttribute[]
  couponProducts CouponProduct[]

  @@index([sku])
  @@index([name])
  @@index([slug])
  @@index([status])
  @@index([isFeatured])
  @@index([price])
  @@map("products")
}

enum ProductStatus {
  ACTIVE
  INACTIVE
  OUT_OF_STOCK
}

model ProductImage {
  id        Int      @id @default(autoincrement())
  productId Int
  url       String
  alt       String?
  title     String?
  order     Int      @default(0)
  isMain    Boolean  @default(false)
  createdAt DateTime @default(now())

  product   Product  @relation(fields: [productId], references: [id], onDelete:
Cascade)

  @@index([productId])
```

```
  @@index([isMain])
  @@map("product_images")
}

model ProductCategory {
  id         Int      @id @default(autoincrement())
  productId  Int
  categoryId Int
  isMain     Boolean  @default(false)

  product    Product  @relation(fields: [productId], references: [id], onDelete:
Cascade)
  category   Category @relation(fields: [categoryId], references: [id], onDelete:
Cascade)

  @@unique([productId, categoryId])
  @@index([productId])
  @@index([categoryId])
  @@map("product_categories")
}

model Attribute {
  id         Int      @id @default(autoincrement())
  name       String
  type       AttributeType
  description String?
  createdAt  DateTime @default(now())
  updatedAt  DateTime @updatedAt

  values     AttributeValue[]
  productAttributes ProductAttribute[]
  variantAttributes VariantAttribute[]

  @@index([name])
  @@map("attributes")
}

enum AttributeType {
  TEXT
  NUMBER
  BOOLEAN
  SELECT
}

model AttributeValue {
  id         Int      @id @default(autoincrement())
  attributeId Int
  value      String
  order      Int      @default(0)

  attribute  Attribute @relation(fields: [attributeId], references: [id], onDelete:
Cascade)
```

```
  productAttributes ProductAttribute[]
  variantAttributes VariantAttribute[]

  @@index([attributeId])
  @@map("attribute_values")
}

model ProductAttribute {
  id          Int      @id @default(autoincrement())
  productId   Int
  attributeId Int
  valueId     Int?
  textValue   String?
  numberValue Decimal?  @db.Decimal(10, 2)
  booleanValue Boolean?

  product     Product   @relation(fields: [productId], references: [id], onDelete:
Cascade)
  attribute   Attribute @relation(fields: [attributeId], references: [id], onDelete:
Cascade)
  value       AttributeValue? @relation(fields: [valueId], references: [id], onDelete:
SetNull)

  @@index([productId])
  @@index([attributeId])
  @@map("product_attributes")
}

model ProductVariant {
  id          Int      @id @default(autoincrement())
  productId   Int
  sku         String   @unique
  price       Decimal?  @db.Decimal(10, 2)
  promotionalPrice Decimal? @db.Decimal(10, 2)
  stock       Int      @default(0)
  image       String?
  status      ProductStatus @default(ACTIVE)
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  product     Product   @relation(fields: [productId], references: [id], onDelete:
Cascade)
  attributes  VariantAttribute[]
  orderItems  OrderItem[]
  cartItems   Cart[]

  @@index([productId])
  @@index([sku])
  @@map("product_variants")
}

model VariantAttribute {
```

```
  id         Int      @id @default(autoincrement())
  variantId  Int
  attributeId Int
  valueId    Int

  variant    ProductVariant @relation(fields: [variantId], references: [id], onDelete:
Cascade)
  attribute  Attribute @relation(fields: [attributeId], references: [id], onDelete:
Cascade)
  value      AttributeValue @relation(fields: [valueId], references: [id], onDelete:
Cascade)

  @@index([variantId])
  @@index([attributeId])
  @@map("variant_attributes")
}

model Order {
  id             Int      @id @default(autoincrement())
  code           String   @unique
  userId         Int?
  shippingAddressId Int?
  billingAddressId  Int?
  status         OrderStatus @default(AWAITING_PAYMENT)
  subtotal       Decimal  @db.Decimal(10, 2)
  discount       Decimal  @default(0) @db.Decimal(10, 2)
  shipping       Decimal  @default(0) @db.Decimal(10, 2)
  total          Decimal  @db.Decimal(10, 2)
  shippingMethod   String?
  trackingCode     String?
  notes          String?  @db.Text
  adminNotes       String?  @db.Text
  customerIp       String?
  userAgent        String?  @db.Text
  createdAt        DateTime  @default(now())
  updatedAt        DateTime  @updatedAt
  paidAt         DateTime?
  shippedAt        DateTime?
  deliveredAt      DateTime?
  canceledAt       DateTime?
  couponId         Int?

  user           User?    @relation(fields: [userId], references: [id], onDelete: SetNull)
  shippingAddress  Address?  @relation("ShippingAddress", fields:
[shippingAddressId], references: [id], onDelete: SetNull)
  billingAddress   Address?  @relation("BillingAddress", fields: [billingAddressId],
references: [id], onDelete: SetNull)
  coupon         Coupon?  @relation(fields: [couponId], references: [id], onDelete:
SetNull)
  items          OrderItem[]
  payments         Payment[]
```

```
  @@index([code])
  @@index([userId])
  @@index([status])
  @@index([createdAt])
  @@map("orders")
}

enum OrderStatus {
  AWAITING_PAYMENT
  PAYMENT_APPROVED
  IN_PREPARATION
  SHIPPED
  DELIVERED
  CANCELED
  RETURNED
}

model OrderItem {
  id          Int      @id @default(autoincrement())
  orderId     Int
  productId   Int?
  variantId   Int?
  productName String
  sku         String
  quantity    Int
  unitPrice   Decimal   @db.Decimal(10, 2)
  subtotal    Decimal   @db.Decimal(10, 2)
  attributes  Json?

  order       Order     @relation(fields: [orderId], references: [id], onDelete: Cascade)
  product     Product?  @relation(fields: [productId], references: [id], onDelete:
SetNull)
  variant     ProductVariant? @relation(fields: [variantId], references: [id], onDelete:
SetNull)

  @@index([orderId])
  @@index([productId])
  @@map("order_items")
}

model Payment {
  id          Int      @id @default(autoincrement())
  orderId     Int
  transactionId String?
  method      PaymentMethod
  status      PaymentStatus @default(PENDING)
  amount      Decimal   @db.Decimal(10, 2)
  installments Int      @default(1)
  gateway     String   @default("mercadopago")
  paymentData Json?
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt
```

```
  order      Order    @relation(fields: [orderId], references: [id], onDelete: Cascade)

  @@index([orderId])
  @@index([transactionId])
  @@index([status])
  @@map("payments")
}

enum PaymentMethod {
  CREDIT_CARD
  BOLETO
  PIX
  BANK_TRANSFER
  MERCADOPAGO
}

enum PaymentStatus {
  PENDING
  APPROVED
  REJECTED
  REFUNDED
  IN_PROCESS
}

model Coupon {
  id          Int      @id @default(autoincrement())
  code        String   @unique
  type        CouponType
  value       Decimal? @db.Decimal(10, 2)
  startDate   DateTime
  endDate     DateTime
  maxUses     Int?
  currentUses Int      @default(0)
  minValue    Decimal? @db.Decimal(10, 2)
  firstPurchase Boolean @default(false)
  status      CouponStatus @default(ACTIVE)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  categories  CouponCategory[]
  products    CouponProduct[]
  orders      Order[]

  @@index([code])
  @@index([status])
  @@index([endDate])
  @@map("coupons")
}

enum CouponType {
  PERCENTAGE
```

```
  FIXED_AMOUNT
  FREE_SHIPPING
}

enum CouponStatus {
  ACTIVE
  INACTIVE
}

model CouponCategory {
  id         Int      @id @default(autoincrement())
  couponId   Int
  categoryId Int

  coupon     Coupon   @relation(fields: [couponId], references: [id], onDelete:
Cascade)
  category   Category @relation(fields: [categoryId], references: [id], onDelete:
Cascade)

  @@unique([couponId, categoryId])
  @@map("coupon_categories")
}

model CouponProduct {
  id         Int      @id @default(autoincrement())
  couponId   Int
  productId  Int

  coupon     Coupon   @relation(fields: [couponId], references: [id], onDelete:
Cascade)
  product    Product  @relation(fields: [productId], references: [id], onDelete:
Cascade)

  @@unique([couponId, productId])
  @@map("coupon_products")
}

model Review {
  id         Int      @id @default(autoincrement())
  productId  Int
  userId     Int?
  name       String
  email      String
  title      String?
  comment    String?  @db.Text
  rating     Int
  status     ReviewStatus @default(PENDING)
  verifiedPurchase Boolean @default(false)
  createdAt  DateTime @default(now())
  approvedAt DateTime?
  adminReply String?  @db.Text
  replyDate  DateTime?
```

```
  product    Product   @relation(fields: [productId], references: [id], onDelete:
Cascade)
  user       User?     @relation(fields: [userId], references: [id], onDelete: SetNull)

  @@index([productId])
  @@index([userId])
  @@index([status])
  @@map("reviews")
}

enum ReviewStatus {
  PENDING
  APPROVED
  REJECTED
}

model Wishlist {
  id         Int      @id @default(autoincrement())
  userId     Int
  productId  Int
  addedAt    DateTime  @default(now())

  user       User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  product    Product   @relation(fields: [productId], references: [id], onDelete:
Cascade)

  @@unique([userId, productId])
  @@index([userId])
  @@map("wishlist")
}

model Cart {
  id         Int      @id @default(autoincrement())
  userId     Int?
  sessionId  String?
  productId  Int
  variantId  Int?
  quantity   Int      @default(1)
  addedAt    DateTime  @default(now())
  updatedAt  DateTime  @updatedAt

  user       User?    @relation(fields: [userId], references: [id], onDelete: Cascade)
  product    Product   @relation(fields: [productId], references: [id], onDelete:
Cascade)
  variant    ProductVariant? @relation(fields: [variantId], references: [id], onDelete:
Cascade)

  @@index([userId])
  @@index([sessionId])
  @@map("cart")
}
```

```
model Setting {
  id          Int      @id @default(autoincrement())
  key         String   @unique
  value       String?  @db.Text
  type        SettingType @default(TEXT)
  group       String   @default("general")
  description String?
  updatedAt   DateTime  @updatedAt

  @@index([key])
  @@index([group])
  @@map("settings")
}

enum SettingType {
  TEXT
  NUMBER
  BOOLEAN
  JSON
}

model Log {
  id          Int      @id @default(autoincrement())
  userId      Int?
  type        String
  action      String
  description String?  @db.Text
  data        Json?
  ip          String?
  userAgent   String?  @db.Text
  createdAt   DateTime  @default(now())

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

  @@index([userId])
  @@index([type])
  @@index([createdAt])
  @@map("logs")
}
```

Autenticação e Autorização

Utilizaremos NextAuth.js para gerenciar autenticação e autorização:

```
// /lib/auth/auth.js
import { PrismaAdapter } from "@next-auth/prisma-adapter";
import { compare } from "bcryptjs";
import NextAuth from "next-auth";
```

```
import CredentialsProvider from "next-auth/providers/credentials";
import prisma from "../prisma";

export const authOptions = {
  adapter: PrismaAdapter(prisma),
  providers: [
    CredentialsProvider({
      name: "Credentials",
      credentials: {
        email: { label: "Email", type: "email" },
        password: { label: "Senha", type: "password" }
      },
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) {
          return null;
        }

        const user = await prisma.user.findUnique({
          where: {
            email: credentials.email
          }
        });

        if (!user || !(await compare(credentials.password, user.password))) {
          return null;
        }

        if (user.status !== "ACTIVE") {
          throw new Error("Conta inativa ou bloqueada");
        }

        // Atualizar último login
        await prisma.user.update({
          where: { id: user.id },
          data: { lastLogin: new Date() }
        });

        return {
          id: user.id.toString(),
          email: user.email,
          name: user.name,
          type: user.type
        };
      }
    })
  ],
  callbacks: {
    async jwt({ token, user }) {
      if (user) {
        token.id = user.id;
        token.type = user.type;
      }
```

```
      return token;
    },
    async session({ session, token }) {
      if (token) {
        session.user.id = token.id;
        session.user.type = token.type;
      }
      return session;
    }
  },
  pages: {
    signIn: "/login",
    error: "/login",
  },
  session: {
    strategy: "jwt",
    maxAge: 30 * 24 * 60 * 60, // 30 dias
  },
  secret: process.env.NEXTAUTH_SECRET,
};

const handler = NextAuth(authOptions);

export { handler as GET, handler as POST };
```

Middleware para proteção de rotas:

```
 // /middleware.js
import { NextResponse } from "next/server";
import { getToken } from "next-auth/jwt";

export async function middleware(request) {
  const token = await getToken({ req: request });
  const isAuthenticated = !!token;
  const isAdmin = token?.type === "ADMIN";
  const isAdminRoute = request.nextUrl.pathname.startsWith("/admin");
  const isApiRoute = request.nextUrl.pathname.startsWith("/api");

  // Proteger rotas administrativas
  if (isAdminRoute && (!isAuthenticated || !isAdmin)) {
    return NextResponse.redirect(new URL("/login", request.url));
  }

  // Proteger APIs administrativas
  if (isApiRoute && request.nextUrl.pathname.startsWith("/api/admin") && (!
isAuthenticated || !isAdmin)) {
    return new NextResponse(
      JSON.stringify({ error: "Acesso não autorizado" }),
      { status: 401, headers: { "Content-Type": "application/json" } }
    );
```

```
  }

  // Proteger APIs que requerem autenticação
  if (isApiRoute &&
      (request.nextUrl.pathname.startsWith("/api/users/me") ||
       request.nextUrl.pathname.startsWith("/api/orders") ||
       request.nextUrl.pathname.startsWith("/api/checkout")) &&
      !isAuthenticated) {
    return new NextResponse(
      JSON.stringify({ error: "Autenticação necessária" }),
      { status: 401, headers: { "Content-Type": "application/json" } }
    );
  }

  return NextResponse.next();
}

export const config = {
  matcher: [
    "/admin/:path*",
    "/api/admin/:path*",
    "/api/users/me/:path*",
    "/api/orders/:path*",
    "/api/checkout/:path*"
  ],
};
```

Implementação das APIs

API de Produtos

```
// /app/api/products/route.js
import { NextResponse } from "next/server";
import prisma from "@/lib/prisma";

export async function GET(request) {
  try {
    const { searchParams } = new URL(request.url);
    const page = parseInt(searchParams.get("page") || "1");
    const limit = parseInt(searchParams.get("limit") || "10");
    const category = searchParams.get("category");
    const search = searchParams.get("search");
    const sort = searchParams.get("sort") || "createdAt";
    const order = searchParams.get("order") || "desc";

    const skip = (page - 1) * limit;

    // Construir filtros
```

```
const where = {
  status: "ACTIVE",
};

if (category) {
  where.categories = {
    some: {
      category: {
        slug: category
      }
    }
  };
}

if (search) {
  where.OR = [
    { name: { contains: search } },
    { shortDescription: { contains: search } },
    { sku: { contains: search } }
  ];
}

// Construir ordenação
const orderBy = {};
orderBy[sort] = order.toLowerCase();

// Buscar produtos
const [products, total] = await Promise.all([
  prisma.product.findMany({
    where,
    orderBy,
    skip,
    take: limit,
    include: {
      images: {
        where: { isMain: true },
        take: 1
      },
      categories: {
        include: {
          category: {
            select: {
              id: true,
              name: true,
              slug: true
            }
          }
        }
      },
      reviews: {
        where: { status: "APPROVED" },
        select: { rating: true }
```

```
        }
      }
    }),
    prisma.product.count({ where })
  ]);

  // Calcular média de avaliações
  const productsWithRating = products.map(product => {
    const totalRatings = product.reviews.length;
    const averageRating = totalRatings > 0
      ? product.reviews.reduce((sum, review) => sum + review.rating, 0) / totalRatings
      : 0;

    return {
      ...product,
      rating: averageRating,
      rating_count: totalRatings,
      reviews: undefined, // Remover array de reviews
      main_image: product.images[0]?.url || null,
      images: undefined, // Remover array de imagens
      categories: product.categories.map(pc => pc.category)
    };
  });

  return NextResponse.json({
    products: productsWithRating,
    pagination: {
      total,
      page,
      limit,
      pages: Math.ceil(total / limit)
    }
  });
 } catch (error) {
  console.error("Error fetching products:", error);
  return NextResponse.json(
    { error: "Erro ao buscar produtos" },
    { status: 500 }
  );
 }
}

export async function POST(request) {
 try {
  const session = await getServerSession(authOptions);

  // Verificar se usuário é admin
  if (!session || session.user.type !== "ADMIN") {
    return NextResponse.json(
      { error: "Não autorizado" },
      { status: 401 }
    );
```

```
}

const data = await request.json();

// Validar dados
const { error } = productSchema.validate(data);
if (error) {
  return NextResponse.json(
    { error: error.details[0].message },
    { status: 400 }
  );
}

// Criar produto
const product = await prisma.product.create({
  data: {
    sku: data.sku,
    name: data.name,
    shortDescription: data.shortDescription,
    longDescription: data.longDescription,
    price: data.price,
    promotionalPrice: data.promotionalPrice,
    stock: data.stock,
    weight: data.weight,
    height: data.height,
    width: data.width,
    depth: data.depth,
    isFeatured: data.isFeatured,
    isNew: data.isNew,
    status: data.status,
    manageStock: data.manageStock,
    allowOutOfStock: data.allowOutOfStock,
    slug: data.slug,
    metaTitle: data.metaTitle,
    metaDescription: data.metaDescription,
    categories: {
      create: data.categories.map(categoryId => ({
        category: {
          connect: { id: categoryId }
        },
        isMain: categoryId === data.mainCategory
      }))
    },
    images: {
      create: data.images.map((image, index) => ({
        url: image.url,
        alt: image.alt || data.name,
        title: image.title,
        order: index,
        isMain: index === 0
      }))
    }
```

```
      }
    });

    return NextResponse.json(product, { status: 201 });
  } catch (error) {
    console.error("Error creating product:", error);
    return NextResponse.json(
      { error: "Erro ao criar produto" },
      { status: 500 }
    );
  }
}
```

API de Detalhes do Produto

```
// /app/api/products/[id]/route.js
import { NextResponse } from "next/server";
import prisma from "@/lib/prisma";

export async function GET(request, { params }) {
  try {
    const id = parseInt(params.id);

    if (isNaN(id)) {
      return NextResponse.json(
        { error: "ID inválido" },
        { status: 400 }
      );
    }

    // Buscar produto com detalhes
    const product = await prisma.product.findUnique({
      where: { id },
      include: {
        images: {
          orderBy: { order: "asc" }
        },
        categories: {
          include: {
            category: {
              select: {
                id: true,
                name: true,
                slug: true
              }
            }
          }
        },
        variants: {
          include: {
```

```
        attributes: {
          include: {
            attribute: true,
            value: true
          }
        }
      }
    },
    productAttributes: {
      include: {
        attribute: true,
        value: true
      }
    },
    reviews: {
      where: { status: "APPROVED" },
      take: 5,
      orderBy: { createdAt: "desc" },
      select: {
        id: true,
        name: true,
        title: true,
        comment: true,
        rating: true,
        createdAt: true,
        verifiedPurchase: true
      }
    }
  }
});

if (!product) {
  return NextResponse.json(
    { error: "Produto não encontrado" },
    { status: 404 }
  );
}

// Incrementar visualizações
await prisma.product.update({
  where: { id },
  data: { views: { increment: 1 } }
});

// Calcular média de avaliações
const totalRatings = await prisma.review.count({
  where: {
    productId: id,
    status: "APPROVED"
  }
});
```

```
    const averageRating = totalRatings > 0
      ? await prisma.review.aggregate({
          where: {
            productId: id,
            status: "APPROVED"
          },
          _avg: {
            rating: true
          }
        }).then(result => result._avg.rating)
      : 0;

    // Formatar resposta
    const formattedProduct = {
      ...product,
      categories: product.categories.map(pc => pc.category),
      rating: averageRating,
      rating_count: totalRatings,
      variants: product.variants.map(variant => ({
        ...variant,
        attributes: variant.attributes.map(attr => ({
          name: attr.attribute.name,
          value: attr.value.value,
          label: attr.value.value
        }))
      })),
      attributes: product.productAttributes.map(attr => ({
        name: attr.attribute.name,
        value: attr.value?.value || attr.textValue || attr.numberValue?.toString() ||
(attr.booleanValue ? "Sim" : "Não")
      })),
      specifications: product.productAttributes.map(attr => ({
        name: attr.attribute.name,
        value: attr.value?.value || attr.textValue || attr.numberValue?.toString() ||
(attr.booleanValue ? "Sim" : "Não")
      }))
    };

    return NextResponse.json(formattedProduct);
  } catch (error) {
    console.error("Error fetching product:", error);
    return NextResponse.json(
      { error: "Erro ao buscar produto" },
      { status: 500 }
    );
  }
}

export async function PUT(request, { params }) {
  try {
    const session = await getServerSession(authOptions);
```

```
    // Verificar se usuário é admin
    if (!session || session.user.type !== "ADMIN") {
      return NextResponse.json(
        { error: "Não autorizado" },
        { status: 401 }
      );
    }

    const id = parseInt(params.id);

    if (isNaN(id)) {
      return NextResponse.json(
        { error: "ID inválido" },
        { status: 400 }
      );
    }

    const data = await request.json();

    // Validar dados
    const { error } = productSchema.validate(data);
    if (error) {
      return NextResponse.json(
        { error: error.details[0].message },
        { status: 400 }
      );
    }

    // Verificar se produto existe
    const existingProduct = await prisma.product.findUnique({
      where: { id }
    });

    if (!existingProduct) {
      return NextResponse.json(
        { error: "Produto não encontrado" },
        { status: 404 }
      );
    }

    // Atualizar produto
    const product = await prisma.product.update({
      where: { id },
      data: {
        sku: data.sku,
        name: data.name,
        shortDescription: data.shortDescription,
        longDescription: data.longDescription,
        price: data.price,
        promotionalPrice: data.promotionalPrice,
        stock: data.stock,
        weight: data.weight,
```

```
      height: data.height,
      width: data.width,
      depth: data.depth,
      isFeatured: data.isFeatured,
      isNew: data.isNew,
      status: data.status,
      manageStock: data.manageStock,
      allowOutOfStock: data.allowOutOfStock,
      slug: data.slug,
      metaTitle: data.metaTitle,
      metaDescription: data.metaDescription,
    }
  });

  // Atualizar categorias
  await prisma.productCategory.deleteMany({
    where: { productId: id }
  });

  await prisma.productCategory.createMany({
    data: data.categories.map(categoryId => ({
      productId: id,
      categoryId,
      isMain: categoryId === data.mainCategory
    }))
  });

  // Atualizar imagens
  if (data.images) {
    await prisma.productImage.deleteMany({
      where: { productId: id }
    });

    await prisma.productImage.createMany({
      data: data.images.map((image, index) => ({
        productId: id,
        url: image.url,
        alt: image.alt || data.name,
        title: image.title,
        order: index,
        isMain: index === 0
      }))
    });
  }

  return NextResponse.json(product);
} catch (error) {
  console.error("Error updating product:", error);
  return NextResponse.json(
    { error: "Erro ao atualizar produto" },
    { status: 500 }
  );
```

```
  }
}

export async function DELETE(request, { params }) {
  try {
    const session = await getServerSession(authOptions);

    // Verificar se usuário é admin
    if (!session || session.user.type !== "ADMIN") {
      return NextResponse.json(
        { error: "Não autorizado" },
        { status: 401 }
      );
    }

    const id = parseInt(params.id);

    if (isNaN(id)) {
      return NextResponse.json(
        { error: "ID inválido" },
        { status: 400 }
      );
    }

    // Verificar se produto existe
    const existingProduct = await prisma.product.findUnique({
      where: { id }
    });

    if (!existingProduct) {
      return NextResponse.json(
        { error: "Produto não encontrado" },
        { status: 404 }
      );
    }

    // Excluir produto
    await prisma.product.delete({
      where: { id }
    });

    return NextResponse.json({ success: true });
  } catch (error) {
    console.error("Error deleting product:", error);
    return NextResponse.json(
      { error: "Erro ao excluir produto" },
      { status: 500 }
    );
  }
}
```

API de Carrinho

```
 // /app/api/cart/route.js
import { NextResponse } from "next/server";
import { getServerSession } from "next-auth";
import { authOptions } from "@/lib/auth/auth";
import prisma from "@/lib/prisma";

export async function GET(request) {
  try {
    const session = await getServerSession(authOptions);
    const { searchParams } = new URL(request.url);
    const sessionId = searchParams.get("session_id");

    let where = {};

    if (session) {
      // Usuário autenticado
      where.userId = parseInt(session.user.id);
    } else if (sessionId) {
      // Usuário não autenticado com ID de sessão
      where.sessionId = sessionId;
    } else {
      // Sem identificação
      return NextResponse.json({ items: [], total: 0 });
    }

    // Buscar itens do carrinho
    const cartItems = await prisma.cart.findMany({
      where,
      include: {
        product: {
          select: {
            id: true,
            name: true,
            slug: true,
            price: true,
            promotionalPrice: true,
            stock: true,
            images: {
              where: { isMain: true },
              take: 1,
              select: { url: true }
            }
          }
        },
        variant: {
          select: {
            id: true,
            sku: true,
            price: true,
```

```
            promotionalPrice: true,
            stock: true,
            image: true,
            attributes: {
              include: {
                attribute: true,
                value: true
              }
            }
          }
        }
      }
    });

    // Formatar itens do carrinho
    const formattedItems = cartItems.map(item => {
      const price = item.variant?.price || item.variant?.promotionalPrice ||
                    item.product.promotionalPrice || item.product.price;

      return {
        id: item.id,
        product_id: item.productId,
        variant_id: item.variantId,
        name: item.product.name,
        slug: item.product.slug,
        price,
        quantity: item.quantity,
        subtotal: parseFloat(price) * item.quantity,
        image: item.variant?.image || item.product.images[0]?.url || null,
        attributes: item.variant?.attributes.map(attr => ({
          name: attr.attribute.name,
          value: attr.value.value
        })) || [],
        stock: item.variant?.stock || item.product.stock
      };
    });

    // Calcular total
    const total = formattedItems.reduce((sum, item) => sum + item.subtotal, 0);

    return NextResponse.json({
      items: formattedItems,
      total
    });
  } catch (error) {
    console.error("Error fetching cart:", error);
    return NextResponse.json(
      { error: "Erro ao buscar carrinho" },
      { status: 500 }
    );
  }
}
```

```
export async function DELETE(request) {
  try {
    const session = await getServerSession(authOptions);
    const { searchParams } = new URL(request.url);
    const sessionId = searchParams.get("session_id");

    let where = {};

    if (session) {
      // Usuário autenticado
      where.userId = parseInt(session.user.id);
    } else if (sessionId) {
      // Usuário não autenticado com ID de sessão
      where.sessionId = sessionId;
    } else {
      // Sem identificação
      return NextResponse.json({ success: false, error: "Identificação necessária" }, {
status: 400 });
    }

    // Limpar carrinho
    await prisma.cart.deleteMany({ where });

    return NextResponse.json({ success: true });
  } catch (error) {
    console.error("Error clearing cart:", error);
    return NextResponse.json(
      { error: "Erro ao limpar carrinho" },
      { status: 500 }
    );
  }
}
```

API de Itens do Carrinho

```
 // /app/api/cart/items/route.js
import { NextResponse } from "next/server";
import { getServerSession } from "next-auth";
import { authOptions } from "@/lib/auth/auth";
import prisma from "@/lib/prisma";

export async function POST(request) {
  try {
    const session = await getServerSession(authOptions);
    const { searchParams } = new URL(request.url);
    const sessionId = searchParams.get("session_id") || `guest_${Date.now()}`;

    const data = await request.json();
```

```
    // Validar dados
    if (!data.product_id || !data.quantity || data.quantity < 1) {
      return NextResponse.json(
        { error: "Dados inválidos" },
        { status: 400 }
      );
    }

    // Verificar se produto existe
    const product = await prisma.product.findUnique({
      where: { id: data.product_id },
      select: {
        id: true,
        stock: true,
        status: true,
        manageStock: true,
        allowOutOfStock: true
      }
    });

    if (!product) {
      return NextResponse.json(
        { error: "Produto não encontrado" },
        { status: 404 }
      );
    }

    // Verificar estoque
    if (product.manageStock && !product.allowOutOfStock && product.stock <
data.quantity) {
      return NextResponse.json(
        { error: "Quantidade indisponível em estoque" },
        { status: 400 }
      );
    }

    // Verificar variante se fornecida
    let variant = null;
    if (data.variant_id) {
      variant = await prisma.productVariant.findUnique({
        where: { id: data.variant_id },
        select: {
          id: true,
          stock: true,
          status: true
        }
      });

      if (!variant) {
        return NextResponse.json(
          { error: "Variante não encontrada" },
          { status: 404 }
```

```
    );
  }

  if (variant.status !== "ACTIVE" || variant.stock < data.quantity) {
    return NextResponse.json(
      { error: "Variante indisponível ou sem estoque suficiente" },
      { status: 400 }
    );
  }
}

// Preparar dados do carrinho
const cartData = {
  productId: data.product_id,
  variantId: data.variant_id || null,
  quantity: data.quantity
};

if (session) {
  // Usuário autenticado
  cartData.userId = parseInt(session.user.id);
} else {
  // Usuário não autenticado
  cartData.sessionId = sessionId;
}

// Verificar se item já existe no carrinho
const existingItem = await prisma.cart.findFirst({
  where: {
    productId: data.product_id,
    variantId: data.variant_id || null,
    ...(session ? { userId: parseInt(session.user.id) } : { sessionId })
  }
});

let cartItem;

if (existingItem) {
  // Atualizar quantidade
  cartItem = await prisma.cart.update({
    where: { id: existingItem.id },
    data: { quantity: existingItem.quantity + data.quantity }
  });
} else {
  // Adicionar novo item
  cartItem = await prisma.cart.create({
    data: cartData
  });
}

// Buscar carrinho atualizado
const response = await fetch(`${request.nextUrl.origin}/api/cart${session ? "" : `?
```

```
session_id=${sessionId}`}`);
    const cart = await response.json();

    return NextResponse.json({
      ...cart,
      session_id: session ? undefined : sessionId
    });
  } catch (error) {
    console.error("Error adding to cart:", error);
    return NextResponse.json(
      { error: "Erro ao adicionar ao carrinho" },
      { status: 500 }
    );
  }
}
```

API de Checkout

```
 // /app/api/checkout/process/route.js
import { NextResponse } from "next/server";
import { getServerSession } from "next-auth";
import { authOptions } from "@/lib/auth/auth";
import prisma from "@/lib/prisma";
import { createPreference } from "@/lib/mercadopago";

export async function POST(request) {
  try {
    const session = await getServerSession(authOptions);

    // Verificar se usuário está autenticado
    if (!session) {
      return NextResponse.json(
        { error: "Autenticação necessária" },
        { status: 401 }
      );
    }

    const data = await request.json();

    // Validar dados
    if (!data.shipping_address || !data.billing_address || !data.payment_method) {
      return NextResponse.json(
        { error: "Dados incompletos" },
        { status: 400 }
      );
    }

    // Buscar itens do carrinho
    const cartItems = await prisma.cart.findMany({
      where: { userId: parseInt(session.user.id) },
```

```
    include: {
      product: {
        select: {
          id: true,
          name: true,
          sku: true,
          price: true,
          promotionalPrice: true,
          stock: true,
          manageStock: true,
          allowOutOfStock: true
        }
      },
      variant: {
        select: {
          id: true,
          sku: true,
          price: true,
          promotionalPrice: true,
          stock: true
        }
      }
    }
  });

  if (!cartItems.length) {
    return NextResponse.json(
      { error: "Carrinho vazio" },
      { status: 400 }
    );
  }

  // Verificar estoque
  for (const item of cartItems) {
    const stock = item.variant?.stock ?? item.product.stock;
    const manageStock = item.product.manageStock;
    const allowOutOfStock = item.product.allowOutOfStock;

    if (manageStock && !allowOutOfStock && stock < item.quantity) {
      return NextResponse.json(
        {
          error: "Produto sem estoque suficiente",
          product: item.product.name,
          available: stock
        },
        { status: 400 }
      );
    }
  }

  // Calcular valores
  const subtotal = cartItems.reduce((sum, item) => {
```

```
    const price = item.variant?.price || item.variant?.promotionalPrice ||
                  item.product.promotionalPrice || item.product.price;
    return sum + (parseFloat(price) * item.quantity);
  }, 0);

  // Aplicar cupom se fornecido
  let discount = 0;
  let couponId = null;

  if (data.coupon_code) {
    const coupon = await prisma.coupon.findFirst({
      where: {
        code: data.coupon_code,
        status: "ACTIVE",
        startDate: { lte: new Date() },
        endDate: { gte: new Date() },
        OR: [
          { maxUses: null },
          { currentUses: { lt: { maxUses: true } } }
        ]
      }
    });

    if (coupon) {
      if (coupon.firstPurchase) {
        const previousOrders = await prisma.order.count({
          where: { userId: parseInt(session.user.id) }
        });

        if (previousOrders > 0) {
          return NextResponse.json(
            { error: "Cupom válido apenas para primeira compra" },
            { status: 400 }
          );
        }
      }

      if (coupon.minValue && subtotal < parseFloat(coupon.minValue)) {
        return NextResponse.json(
          {
            error: "Valor mínimo não atingido para o cupom",
            minValue: parseFloat(coupon.minValue)
          },
          { status: 400 }
        );
      }

      if (coupon.type === "PERCENTAGE") {
        discount = subtotal * (parseFloat(coupon.value) / 100);
      } else if (coupon.type === "FIXED_AMOUNT") {
        discount = parseFloat(coupon.value);
      }
```

```
    couponId = coupon.id;
  } else {
    return NextResponse.json(
      { error: "Cupom inválido ou expirado" },
      { status: 400 }
    );
  }
}

// Calcular frete
const shipping = parseFloat(data.shipping_cost || 0);

// Calcular total
const total = subtotal - discount + shipping;

// Criar endereços se necessário
let shippingAddressId = data.shipping_address.id;
let billingAddressId = data.billing_address.id;

if (!shippingAddressId) {
  const shippingAddress = await prisma.address.create({
    data: {
      userId: parseInt(session.user.id),
      name: data.shipping_address.name,
      zipCode: data.shipping_address.zip_code,
      street: data.shipping_address.street,
      number: data.shipping_address.number,
      complement: data.shipping_address.complement,
      neighborhood: data.shipping_address.neighborhood,
      city: data.shipping_address.city,
      state: data.shipping_address.state,
      country: data.shipping_address.country || "Brasil",
      phone: data.shipping_address.phone,
      type: "SHIPPING"
    }
  });

  shippingAddressId = shippingAddress.id;
}

if (!billingAddressId) {
  if (data.same_address) {
    billingAddressId = shippingAddressId;
  } else {
    const billingAddress = await prisma.address.create({
      data: {
        userId: parseInt(session.user.id),
        name: data.billing_address.name,
        zipCode: data.billing_address.zip_code,
        street: data.billing_address.street,
        number: data.billing_address.number,
```

```
          complement: data.billing_address.complement,
          neighborhood: data.billing_address.neighborhood,
          city: data.billing_address.city,
          state: data.billing_address.state,
          country: data.billing_address.country || "Brasil",
          phone: data.billing_address.phone,
          type: "BILLING"
        }
      });

      billingAddressId = billingAddress.id;
    }
  }

  // Gerar código do pedido
  const orderCode = `PED${Date.now()}${Math.floor(Math.random() * 1000)}`;

  // Criar pedido
  const order = await prisma.order.create({
    data: {
      code: orderCode,
      userId: parseInt(session.user.id),
      shippingAddressId,
      billingAddressId,
      status: "AWAITING_PAYMENT",
      subtotal,
      discount,
      shipping,
      total,
      shippingMethod: data.shipping_method,
      notes: data.notes,
      customerIp: request.headers.get("x-forwarded-for") || request.ip,
      userAgent: request.headers.get("user-agent"),
      couponId,
      items: {
        create: cartItems.map(item => ({
          productId: item.productId,
          variantId: item.variantId,
          productName: item.product.name,
          sku: item.variant?.sku || item.product.sku,
          quantity: item.quantity,
          unitPrice: parseFloat(item.variant?.price || item.variant?.promotionalPrice
||
                item.product.promotionalPrice || item.product.price),
          subtotal: parseFloat(item.variant?.price || item.variant?.promotionalPrice
||
              item.product.promotionalPrice || item.product.price) * item.quantity,
          attributes: item.variant ? {
            // Buscar atributos da variante
          } : null
        }))
      }
```

```
  }
});

// Atualizar estoque
for (const item of cartItems) {
  if (item.product.manageStock) {
    if (item.variantId) {
      await prisma.productVariant.update({
        where: { id: item.variantId },
        data: { stock: { decrement: item.quantity } }
      });
    } else {
      await prisma.product.update({
        where: { id: item.productId },
        data: { stock: { decrement: item.quantity } }
      });
    }
  }
}

// Atualizar uso do cupom
if (couponId) {
  await prisma.coupon.update({
    where: { id: couponId },
    data: { currentUses: { increment: 1 } }
  });
}

// Limpar carrinho
await prisma.cart.deleteMany({
  where: { userId: parseInt(session.user.id) }
});

// Criar pagamento
let paymentData = {
  orderId: order.id,
  method: data.payment_method,
  status: "PENDING",
  amount: total,
  installments: data.installments || 1,
  gateway: "mercadopago"
};

// Integração com Mercado Pago
if (data.payment_method === "MERCADOPAGO") {
  const items = cartItems.map(item => ({
    id: item.variant?.sku || item.product.sku,
    title: item.product.name,
    quantity: item.quantity,
    unit_price: parseFloat(item.variant?.price || item.variant?.promotionalPrice ||
          item.product.promotionalPrice || item.product.price)
  }));
```

```
    const preference = await createPreference({
      items,
      external_reference: order.code,
      notification_url: `${process.env.NEXT_PUBLIC_API_URL}/api/webhooks/
mercadopago`,
      back_urls: {
        success: `${process.env.NEXT_PUBLIC_FRONTEND_URL}/checkout/success?
order=${order.code}`,
        failure: `${process.env.NEXT_PUBLIC_FRONTEND_URL}/checkout/failure?
order=${order.code}`,
        pending: `${process.env.NEXT_PUBLIC_FRONTEND_URL}/checkout/pending?
order=${order.code}`
      },
      auto_return: "approved",
      statement_descriptor: process.env.STORE_NAME || "E-commerce",
      payer: {
        name: session.user.name.split(" ")[0],
        surname: session.user.name.split(" ").slice(1).join(" "),
        email: session.user.email,
        phone: {
          area_code: "",
          number: data.billing_address.phone
        },
        address: {
          zip_code: data.billing_address.zip_code,
          street_name: data.billing_address.street,
          street_number: data.billing_address.number
        }
      }
    });

    paymentData.paymentData = preference;
  }

  const payment = await prisma.payment.create({
    data: paymentData
  });

  return NextResponse.json({
    order: {
      id: order.id,
      code: order.code,
      total: order.total,
      status: order.status
    },
    payment: {
      id: payment.id,
      method: payment.method,
      status: payment.status,
      preference: payment.paymentData
    }
```

```
    });
  } catch (error) {
    console.error("Error processing checkout:", error);
    return NextResponse.json(
      { error: "Erro ao processar pedido" },
      { status: 500 }
    );
  }
}
```

Integração com Mercado Pago

```
// /lib/mercadopago/index.js
import { MercadoPagoConfig, Preference } from 'mercadopago';

const client = new MercadoPagoConfig({
  accessToken: process.env.MERCADOPAGO_ACCESS_TOKEN
});

export async function createPreference(data) {
  try {
    const preference = new Preference(client);

    const response = await preference.create({
      items: data.items,
      external_reference: data.external_reference,
      notification_url: data.notification_url,
      back_urls: data.back_urls,
      auto_return: data.auto_return,
      statement_descriptor: data.statement_descriptor,
      payer: data.payer
    });

    return response;
  } catch (error) {
    console.error('Error creating Mercado Pago preference:', error);
    throw error;
  }
}

export async function getPaymentById(paymentId) {
  try {
    const payment = await client.payment.get({ id: paymentId });
    return payment;
  } catch (error) {
    console.error('Error getting Mercado Pago payment:', error);
    throw error;
  }
}
```

```
// /lib/mercadopago/webhook.js
import prisma from '@/lib/prisma';
import { getPaymentById } from './index';

export async function handleWebhook(data) {
  try {
    // Verificar tipo de notificação
    if (data.type !== 'payment') {
      return { success: true, message: 'Notificação ignorada: não é um pagamento' };
    }

    // Buscar detalhes do pagamento
    const paymentInfo = await getPaymentById(data.data.id);

    // Buscar pedido pelo código
    const order = await prisma.order.findFirst({
      where: { code: paymentInfo.external_reference },
      include: { payments: true }
    });

    if (!order) {
      throw new Error(`Pedido não encontrado: ${paymentInfo.external_reference}`);
    }

    // Mapear status do Mercado Pago para status interno
    let paymentStatus;
    let orderStatus;

    switch (paymentInfo.status) {
      case 'approved':
        paymentStatus = 'APPROVED';
        orderStatus = 'PAYMENT_APPROVED';
        break;
      case 'pending':
      case 'in_process':
        paymentStatus = 'PENDING';
        orderStatus = 'AWAITING_PAYMENT';
        break;
      case 'rejected':
        paymentStatus = 'REJECTED';
        orderStatus = 'AWAITING_PAYMENT';
        break;
      case 'refunded':
      case 'cancelled':
        paymentStatus = 'REFUNDED';
        orderStatus = 'CANCELED';
        break;
      default:
        paymentStatus = 'PENDING';
        orderStatus = 'AWAITING_PAYMENT';
    }
```

```
    // Atualizar pagamento
    await prisma.payment.update({
      where: { id: order.payments[0].id },
      data: {
        transactionId: paymentInfo.id.toString(),
        status: paymentStatus,
        paymentData: paymentInfo
      }
    });

    // Atualizar pedido
    const orderUpdate = {
      status: orderStatus
    };

    if (orderStatus === 'PAYMENT_APPROVED') {
      orderUpdate.paidAt = new Date();
    } else if (orderStatus === 'CANCELED') {
      orderUpdate.canceledAt = new Date();
    }

    await prisma.order.update({
      where: { id: order.id },
      data: orderUpdate
    });

    return {
      success: true,
      message: 'Webhook processado com sucesso',
      order: order.code,
      status: orderStatus
    };
  } catch (error) {
    console.error('Error processing Mercado Pago webhook:', error);
    throw error;
  }
}

// /app/api/webhooks/mercadopago/route.js
import { NextResponse } from 'next/server';
import { handleWebhook } from '@/lib/mercadopago/webhook';

export async function POST(request) {
  try {
    const data = await request.json();

    // Processar webhook
    const result = await handleWebhook(data);

    return NextResponse.json(result);
  } catch (error) {
```

```
    console.error('Error in Mercado Pago webhook:', error);
    return NextResponse.json(
      { error: 'Erro ao processar webhook' },
      { status: 500 }
    );
  }
}
```

Serviços

Serviço de Produtos

```
// /services/ProductService.js
import prisma from '@/lib/prisma';

export default class ProductService {
  static async getProducts(options = {}) {
    const {
      page = 1,
      limit = 10,
      category,
      search,
      sort = 'createdAt',
      order = 'desc',
      featured,
      new: isNew,
      onSale
    } = options;

    const skip = (page - 1) * limit;

    // Construir filtros
    const where = {
      status: 'ACTIVE',
    };

    if (category) {
      where.categories = {
        some: {
          category: {
            slug: category
          }
        }
      };
    }

    if (search) {
      where.OR = [
```

```
      { name: { contains: search } },
      { shortDescription: { contains: search } },
      { sku: { contains: search } }
    ];
  }

  if (featured) {
    where.isFeatured = true;
  }

  if (isNew) {
    where.isNew = true;
  }

  if (onSale) {
    where.promotionalPrice = { not: null };
  }

  // Construir ordenação
  const orderBy = {};
  orderBy[sort] = order.toLowerCase();

  // Buscar produtos
  const [products, total] = await Promise.all([
    prisma.product.findMany({
      where,
      orderBy,
      skip,
      take: limit,
      include: {
        images: {
          where: { isMain: true },
          take: 1
        },
        categories: {
          include: {
            category: {
              select: {
                id: true,
                name: true,
                slug: true
              }
            }
          }
        },
        reviews: {
          where: { status: 'APPROVED' },
          select: { rating: true }
        }
      }
    }),
    prisma.product.count({ where })
```

```
    ]);

    // Calcular média de avaliações
    const productsWithRating = products.map(product => {
      const totalRatings = product.reviews.length;
      const averageRating = totalRatings > 0
        ? product.reviews.reduce((sum, review) => sum + review.rating, 0) / totalRatings
        : 0;

      return {
        ...product,
        rating: averageRating,
        rating_count: totalRatings,
        reviews: undefined, // Remover array de reviews
        main_image: product.images[0]?.url || null,
        images: undefined, // Remover array de imagens
        categories: product.categories.map(pc => pc.category)
      };
    });

    return {
      products: productsWithRating,
      pagination: {
        total,
        page,
        limit,
        pages: Math.ceil(total / limit)
      }
    };
  }

  static async getProductById(id) {
    const product = await prisma.product.findUnique({
      where: { id },
      include: {
        images: {
          orderBy: { order: 'asc' }
        },
        categories: {
          include: {
            category: {
              select: {
                id: true,
                name: true,
                slug: true
              }
            }
          }
        },
        variants: {
          include: {
            attributes: {
```

```
          include: {
            attribute: true,
            value: true
          }
        }
      }
    },
    productAttributes: {
      include: {
        attribute: true,
        value: true
      }
    },
    reviews: {
      where: { status: 'APPROVED' },
      take: 5,
      orderBy: { createdAt: 'desc' },
      select: {
        id: true,
        name: true,
        title: true,
        comment: true,
        rating: true,
        createdAt: true,
        verifiedPurchase: true
      }
    }
  }
});

if (!product) {
  return null;
}

// Incrementar visualizações
await prisma.product.update({
  where: { id },
  data: { views: { increment: 1 } }
});

// Calcular média de avaliações
const totalRatings = await prisma.review.count({
  where: {
    productId: id,
    status: 'APPROVED'
  }
});

const averageRating = totalRatings > 0
  ? await prisma.review.aggregate({
      where: {
        productId: id,
```

```
          status: 'APPROVED'
        },
        _avg: {
          rating: true
        }
      }).then(result => result._avg.rating)
    : 0;

  // Formatar resposta
  return {
    ...product,
    categories: product.categories.map(pc => pc.category),
    rating: averageRating,
    rating_count: totalRatings,
    variants: product.variants.map(variant => ({
      ...variant,
      attributes: variant.attributes.map(attr => ({
        name: attr.attribute.name,
        value: attr.value.value,
        label: attr.value.value
      }))
    })),
    attributes: product.productAttributes.map(attr => ({
      name: attr.attribute.name,
      value: attr.value?.value || attr.textValue || attr.numberValue?.toString() ||
(attr.booleanValue ? 'Sim' : 'Não')
    })),
    specifications: product.productAttributes.map(attr => ({
      name: attr.attribute.name,
      value: attr.value?.value || attr.textValue || attr.numberValue?.toString() ||
(attr.booleanValue ? 'Sim' : 'Não')
    }))
  };
 }

 static async getProductBySlug(slug) {
  const product = await prisma.product.findUnique({
    where: { slug },
    include: {
      images: {
        orderBy: { order: 'asc' }
      },
      categories: {
        include: {
          category: {
            select: {
              id: true,
              name: true,
              slug: true
            }
          }
        }
      }
```

```
    },
    variants: {
      include: {
        attributes: {
          include: {
            attribute: true,
            value: true
          }
        }
      }
    },
    productAttributes: {
      include: {
        attribute: true,
        value: true
      }
    },
    reviews: {
      where: { status: 'APPROVED' },
      take: 5,
      orderBy: { createdAt: 'desc' },
      select: {
        id: true,
        name: true,
        title: true,
        comment: true,
        rating: true,
        createdAt: true,
        verifiedPurchase: true
      }
    }
  }
});

if (!product) {
  return null;
}

// Incrementar visualizações
await prisma.product.update({
  where: { id: product.id },
  data: { views: { increment: 1 } }
});

// Calcular média de avaliações
const totalRatings = await prisma.review.count({
  where: {
    productId: product.id,
    status: 'APPROVED'
  }
});
```

```
    const averageRating = totalRatings > 0
      ? await prisma.review.aggregate({
          where: {
            productId: product.id,
            status: 'APPROVED'
          },
          _avg: {
            rating: true
          }
        }).then(result => result._avg.rating)
      : 0;

    // Formatar resposta
    return {
      ...product,
      categories: product.categories.map(pc => pc.category),
      rating: averageRating,
      rating_count: totalRatings,
      variants: product.variants.map(variant => ({
        ...variant,
        attributes: variant.attributes.map(attr => ({
          name: attr.attribute.name,
          value: attr.value.value,
          label: attr.value.value
        }))
      })),
      attributes: product.productAttributes.map(attr => ({
        name: attr.attribute.name,
        value: attr.value?.value || attr.textValue || attr.numberValue?.toString() ||
(attr.booleanValue ? 'Sim' : 'Não')
      })),
      specifications: product.productAttributes.map(attr => ({
        name: attr.attribute.name,
        value: attr.value?.value || attr.textValue || attr.numberValue?.toString() ||
(attr.booleanValue ? 'Sim' : 'Não')
      }))
    };
  }

  static async getFeaturedProducts(limit = 8) {
    return this.getProducts({
      featured: true,
      limit,
      sort: 'createdAt',
      order: 'desc'
    });
  }

  static async getNewProducts(limit = 8) {
    return this.getProducts({
      new: true,
      limit,
```

```
    sort: 'createdAt',
    order: 'desc'
  });
}

static async getOnSaleProducts(limit = 8) {
  return this.getProducts({
    onSale: true,
    limit,
    sort: 'createdAt',
    order: 'desc'
  });
}

static async getRelatedProducts(productId, categoryId, limit = 4) {
  // Buscar produtos da mesma categoria, excluindo o produto atual
  const products = await prisma.product.findMany({
    where: {
      id: { not: productId },
      status: 'ACTIVE',
      categories: {
        some: {
          categoryId
        }
      }
    },
    take: limit,
    include: {
      images: {
        where: { isMain: true },
        take: 1
      },
      reviews: {
        where: { status: 'APPROVED' },
        select: { rating: true }
      }
    }
  });

  // Calcular média de avaliações
  return products.map(product => {
    const totalRatings = product.reviews.length;
    const averageRating = totalRatings > 0
      ? product.reviews.reduce((sum, review) => sum + review.rating, 0) / totalRatings
      : 0;

    return {
      ...product,
      rating: averageRating,
      rating_count: totalRatings,
      reviews: undefined, // Remover array de reviews
      main_image: product.images[0]?.url || null,
```

```
      images: undefined // Remover array de imagens
    };
  });
 }

 static async searchProducts(query, options = {}) {
  return this.getProducts({
   ...options,
   search: query
  });
 }
}
```

Serviço de Pedidos

```
// /services/OrderService.js
import prisma from '@/lib/prisma';

export default class OrderService {
 static async getOrders(userId, options = {}) {
  const {
   page = 1,
   limit = 10,
   status
  } = options;

  const skip = (page - 1) * limit;

  // Construir filtros
  const where = { userId };

  if (status) {
   where.status = status;
  }

  // Buscar pedidos
  const [orders, total] = await Promise.all([
   prisma.order.findMany({
    where,
    orderBy: { createdAt: 'desc' },
    skip,
    take: limit,
    include: {
     items: {
      include: {
       product: {
        select: {
         slug: true
        }
       }
```

```
          }
        },
        payments: {
          select: {
            method: true,
            status: true
          }
        }
      }
    }),
    prisma.order.count({ where })
  ]);

  return {
    orders,
    pagination: {
      total,
      page,
      limit,
      pages: Math.ceil(total / limit)
    }
  };
}

static async getOrderById(id, userId) {
  const order = await prisma.order.findFirst({
    where: {
      id,
      userId
    },
    include: {
      items: {
        include: {
          product: {
            select: {
              slug: true,
              images: {
                where: { isMain: true },
                take: 1,
                select: { url: true }
              }
            }
          }
        }
      },
      payments: true,
      shippingAddress: true,
      billingAddress: true
    }
  });

  if (!order) {
```

```
    return null;
  }

  // Formatar itens
  const formattedItems = order.items.map(item => ({
    ...item,
    product_slug: item.product?.slug,
    product_image: item.product?.images[0]?.url || null,
    product: undefined
  }));

  return {
    ...order,
    items: formattedItems
  };
}

static async getOrderByCode(code, userId) {
  const order = await prisma.order.findFirst({
    where: {
      code,
      userId
    },
    include: {
      items: {
        include: {
          product: {
            select: {
              slug: true,
              images: {
                where: { isMain: true },
                take: 1,
                select: { url: true }
              }
            }
          }
        }
      },
      payments: true,
      shippingAddress: true,
      billingAddress: true
    }
  });

  if (!order) {
    return null;
  }

  // Formatar itens
  const formattedItems = order.items.map(item => ({
    ...item,
    product_slug: item.product?.slug,
```

```
      product_image: item.product?.images[0]?.url || null,
      product: undefined
    }));

    return {
      ...order,
      items: formattedItems
    };
  }

  static async cancelOrder(id, userId) {
    // Verificar se pedido existe e pertence ao usuário
    const order = await prisma.order.findFirst({
      where: {
        id,
        userId
      }
    });

    if (!order) {
      throw new Error('Pedido não encontrado');
    }

    // Verificar se pedido pode ser cancelado
    if (!['AWAITING_PAYMENT', 'PAYMENT_APPROVED'].includes(order.status)) {
      throw new Error('Este pedido não pode ser cancelado');
    }

    // Atualizar pedido
    const updatedOrder = await prisma.order.update({
      where: { id },
      data: {
        status: 'CANCELED',
        canceledAt: new Date()
      }
    });

    // Restaurar estoque
    const orderItems = await prisma.orderItem.findMany({
      where: { orderId: id },
      include: {
        product: {
          select: {
            id: true,
            manageStock: true
          }
        }
      }
    });

    for (const item of orderItems) {
      if (item.product?.manageStock) {
```

```
      if (item.variantId) {
        await prisma.productVariant.update({
          where: { id: item.variantId },
          data: { stock: { increment: item.quantity } }
        });
      } else if (item.productId) {
        await prisma.product.update({
          where: { id: item.productId },
          data: { stock: { increment: item.quantity } }
        });
      }
    }
  }

  return updatedOrder;
}

static async getOrdersForAdmin(options = {}) {
  const {
    page = 1,
    limit = 10,
    status,
    search
  } = options;

  const skip = (page - 1) * limit;

  // Construir filtros
  const where = {};

  if (status) {
    where.status = status;
  }

  if (search) {
    where.OR = [
      { code: { contains: search } },
      { user: { name: { contains: search } } },
      { user: { email: { contains: search } } }
    ];
  }

  // Buscar pedidos
  const [orders, total] = await Promise.all([
    prisma.order.findMany({
      where,
      orderBy: { createdAt: 'desc' },
      skip,
      take: limit,
      include: {
        user: {
          select: {
```

```
          id: true,
          name: true,
          email: true
        }
      },
      payments: {
        select: {
          method: true,
          status: true
        }
      }
    }
  }),
  prisma.order.count({ where })
 ]);

 return {
  orders,
  pagination: {
   total,
   page,
   limit,
   pages: Math.ceil(total / limit)
  }
 };
}

static async updateOrderStatus(id, status, adminId) {
 // Verificar se pedido existe
 const order = await prisma.order.findUnique({
  where: { id }
 });

 if (!order) {
  throw new Error('Pedido não encontrado');
 }

 // Preparar dados de atualização
 const updateData = {
  status
 };

 // Atualizar timestamps específicos baseados no status
 switch (status) {
  case 'PAYMENT_APPROVED':
   updateData.paidAt = new Date();
   break;
  case 'SHIPPED':
   updateData.shippedAt = new Date();
   break;
  case 'DELIVERED':
   updateData.deliveredAt = new Date();
```

```
        break;
      case 'CANCELED':
        updateData.canceledAt = new Date();
        break;
    }

    // Atualizar pedido
    const updatedOrder = await prisma.order.update({
      where: { id },
      data: updateData
    });

    // Registrar log de alteração
    await prisma.log.create({
      data: {
        userId: adminId,
        type: 'ORDER',
        action: 'STATUS_UPDATE',
        description: `Status do pedido ${order.code} alterado de ${order.status} para ${status}`,
        data: {
          orderId: id,
          orderCode: order.code,
          oldStatus: order.status,
          newStatus: status
        }
      }
    });

    return updatedOrder;
  }
}
```

Considerações de Segurança

1. Proteção contra CSRF

Implementaremos proteção contra Cross-Site Request Forgery usando tokens CSRF:

```
// /lib/csrf.js
import { randomBytes } from 'crypto';

export function generateCsrfToken() {
  return randomBytes(32).toString('hex');
}

export function validateCsrfToken(token, storedToken) {
  if (!token || !storedToken) {
```

```
    return false;
  }

  return token === storedToken;
}

// Middleware para verificar token CSRF
export function csrfProtection(handler) {
  return async (req, res) => {
    // Verificar se é uma requisição mutável
    if (['POST', 'PUT', 'DELETE', 'PATCH'].includes(req.method)) {
      const token = req.headers['x-csrf-token'];
      const storedToken = req.cookies['csrf-token'];

      if (!validateCsrfToken(token, storedToken)) {
        return res.status(403).json({ error: 'Invalid CSRF token' });
      }
    }

    return handler(req, res);
  };
}
```

2. Validação de Dados

Utilizaremos Zod para validação de dados:

```
// /lib/validators/product.js
import { z } from 'zod';

export const productSchema = z.object({
  sku: z.string().min(3).max(50),
  name: z.string().min(3).max(255),
  shortDescription: z.string().max(255).optional(),
  longDescription: z.string().optional(),
  price: z.number().positive(),
  promotionalPrice: z.number().positive().optional(),
  stock: z.number().int().min(0),
  weight: z.number().positive().optional(),
  height: z.number().positive().optional(),
  width: z.number().positive().optional(),
  depth: z.number().positive().optional(),
  isFeatured: z.boolean().default(false),
  isNew: z.boolean().default(false),
  status: z.enum(['ACTIVE', 'INACTIVE', 'OUT_OF_STOCK']).default('ACTIVE'),
  manageStock: z.boolean().default(true),
  allowOutOfStock: z.boolean().default(false),
  slug: z.string().min(3).max(255),
  metaTitle: z.string().max(100).optional(),
  metaDescription: z.string().max(255).optional(),
```

```
  categories: z.array(z.number().int().positive()),
  mainCategory: z.number().int().positive(),
  images: z.array(
    z.object({
      url: z.string().url(),
      alt: z.string().max(100).optional(),
      title: z.string().max(100).optional()
    })
  ).min(1)
});
```

3. Proteção contra Injeção SQL

O Prisma ORM já fornece proteção contra injeção SQL por padrão, pois utiliza consultas parametrizadas.

4. Autenticação e Autorização

Utilizamos NextAuth.js para autenticação segura e middleware para proteção de rotas.

5. Rate Limiting

Implementaremos limitação de taxa para prevenir ataques de força bruta:

```
// /lib/rate-limit.js
import { LRUCache } from 'lru-cache';

const rateLimit = {
  tokenCache: new LRUCache({
    max: 500,
    ttl: 60 * 1000 // 1 minuto
  }),

  check: (limit, token) => {
    const tokenCount = rateLimit.tokenCache.get(token) || 0;

    if (tokenCount >= limit) {
      return false;
    }

    rateLimit.tokenCache.set(token, tokenCount + 1);
    return true;
  }
};

export default function rateLimiter(options) {
  return async function(req, res, next) {
    const { limit = 10, token = req.ip } = options;
```

```
    if (!rateLimit.check(limit, token)) {
      return res.status(429).json({ error: 'Too many requests' });
    }

    return next();
  };
}
```

Estratégias de Performance

1. Caching

Implementaremos estratégias de cache para melhorar a performance:

```
// /lib/cache.js
import { LRUCache } from 'lru-cache';

const cache = new LRUCache({
  max: 500,
  ttl: 1000 * 60 * 5 // 5 minutos
});

export function getCachedData(key) {
  return cache.get(key);
}

export function setCachedData(key, data, ttl) {
  cache.set(key, data, { ttl });
  return data;
}

export function invalidateCache(key) {
  cache.delete(key);
}

export function invalidateCachePattern(pattern) {
  const keys = cache.keys();
  for (const key of keys) {
    if (key.includes(pattern)) {
      cache.delete(key);
    }
  }
}

// Middleware para cache de API
export function apiCache(ttl = 1000 * 60 * 5) {
  return async (req, res, next) => {
```

```
    // Apenas cache para GET
    if (req.method !== 'GET') {
      return next();
    }

    const key = `api:${req.url}`;
    const cachedData = getCachedData(key);

    if (cachedData) {
      return res.json(cachedData);
    }

    // Substituir res.json para interceptar e cachear a resposta
    const originalJson = res.json;
    res.json = function(data) {
      setCachedData(key, data, ttl);
      return originalJson.call(this, data);
    };

    return next();
  };
}
```

2. Otimização de Consultas

Utilizaremos índices e consultas otimizadas no Prisma:

```
// Exemplo de consulta otimizada com seleção específica de campos
const products = await prisma.product.findMany({
  where: { status: 'ACTIVE' },
  select: {
    id: true,
    name: true,
    price: true,
    promotionalPrice: true,
    slug: true,
    images: {
      where: { isMain: true },
      take: 1,
      select: { url: true }
    }
  }
});
```

3. Paginação

Implementaremos paginação eficiente para grandes conjuntos de dados:

```
// /lib/utils/pagination.js
export function getPaginationParams(query) {
  const page = parseInt(query.page) || 1;
  const limit = parseInt(query.limit) || 10;
  const skip = (page - 1) * limit;

  return { page, limit, skip };
}

export function createPaginationResponse(items, total, page, limit) {
  return {
    items,
    pagination: {
      total,
      page,
      limit,
      pages: Math.ceil(total / limit)
    }
  };
}
```

Conclusão

O planejamento do backend com Next.js para o e-commerce foi projetado seguindo as melhores práticas de desenvolvimento, com foco em modularidade, segurança, performance e escalabilidade.

A arquitetura baseada em API RESTful, com uso do Prisma ORM para interação com o banco de dados MySQL, proporciona uma base sólida para o desenvolvimento do sistema. A integração com o Mercado Pago para processamento de pagamentos atende ao requisito específico do cliente.

Os componentes e serviços detalhados neste documento cobrem todas as funcionalidades essenciais de um e-commerce completo, incluindo:

1. Gerenciamento de produtos e categorias
2. Carrinho de compras
3. Processamento de pedidos
4. Integração de pagamentos
5. Gerenciamento de usuários
6. Avaliações de produtos
7. Cupons de desconto
8. Cálculo de frete

As considerações de segurança e performance garantirão que a aplicação seja não apenas funcional, mas também segura e rápida, proporcionando uma experiência de usuário de alta qualidade.

Este planejamento serve como um guia abrangente para a implementação do backend do e-commerce, fornecendo uma estrutura clara e detalhada para o desenvolvimento.