Sistema de Controle de Vendas de Joias - Documentação Completa

📋 Índice

Visão Geral do Projeto

O **Sistema de Controle de Vendas de Joias** é uma aplicação web full-stack desenvolvida para automatizar e gerenciar vendas de joias. O sistema permite:

- Registrar vendas com código do produto, cliente, valor, forma de pagamento e data
- Visualizar relatórios com cálculos automáticos de vendas brutas e comissão (30%)
- Editar e excluir vendas registradas
- Filtrar vendas por cliente, forma de pagamento e período de datas
- Exportar relatórios em formato Excel

Stack Tecnológico:

- **Frontend:** React 19 + TypeScript + Tailwind CSS 4

- **Backend:** Express.js + tRPC 11
- **Banco de Dados:** MySQL/TiDB com Drizzle ORM
- **Autenticação:** Manus OAuth
- **Exportação:** XLSX (Excel)

Arquitetura e Tecnologias

Fluxo de Dados

```
Cliente (React)
    ↓
tRPC Client (client/src/lib/trpc.ts)
    ↓
tRPC Procedures (server/routers.ts)
    ↓
Database Helpers (server/db.ts)
    ↓
Drizzle ORM (drizzle/schema.ts)
    ↓
MySQL Database
```

Estrutura de Pastas

```
jewelry_sales_control/
├── client/
│   ├── src/
│   │   ├── pages/
│   │   │   ├── Home.tsx           # Página inicial com menu
│   │   │   ├── SalesForm.tsx      # Formulário de registro
│   │   │   ├── SalesReport.tsx    # Relatórios e filtros
│   │   │   └── SalesManagement.tsx # Gerenciamento (editar/deletar)
│   │   ├── components/            # Componentes reutilizáveis
│   │   ├── lib/trpc.ts            # Cliente tRPC
│   │   ├── App.tsx                # Rotas e layout
│   │   └── index.css              # Estilos globais
│   └── public/                    # Assets estáticos
├── server/
│   ├── routers.ts                 # Procedimentos tRPC
│   ├── db.ts                      # Funções de banco de dados
│   └── _core/                     # Infraestrutura (OAuth, contexto, etc)
├── drizzle/
│   └── schema.ts                  # Definição de tabelas
└── shared/                        # Constantes compartilhadas
```

Banco de Dados

Tabela: users

Gerencia usuários autenticados via Manus OAuth.

```
export const users = mysqlTable("users", {
  id: int("id").autoincrement().primaryKey(),
  openId: varchar("openId", { length: 64 }).notNull().unique(),
  name: text("name"),
  email: varchar("email", { length: 320 }),
  loginMethod: varchar("loginMethod", { length: 64 }),
  role: mysqlEnum("role", ["user", "admin"]).default("user").notNull(),
  createdAt: timestamp("createdAt").defaultNow().notNull(),
  updatedAt: timestamp("updatedAt").defaultNow().onUpdateNow().notNull(),
  lastSignedIn: timestamp("lastSignedIn").defaultNow().notNull(),
});
```

Tabela: sales

Armazena informações de cada venda de joias.

```
export const sales = mysqlTable("sales", {
  id: int("id").autoincrement().primaryKey(),
  userId: int("userId").notNull(),
  productCode: varchar("productCode", { length: 100 }).notNull(),
  clientName: varchar("clientName", { length: 255 }).notNull(),
  value: decimal("value", { precision: 10, scale: 2 }).notNull(),
  paymentMethod: varchar("paymentMethod", { length: 50 }).notNull(),
  paymentDate: timestamp("paymentDate").notNull(),
  createdAt: timestamp("createdAt").defaultNow().notNull(),
  updatedAt: timestamp("updatedAt").defaultNow().onUpdateNow().notNull(),
});
```

Campos:

- `id`: Identificador único da venda
- `userId`: Referência ao usuário que registrou a venda
- `productCode`: Código do produto (ex: “JOI001”)
- `clientName`: Nome do cliente
- `value`: Valor da venda em reais (com 2 casas decimais)
- `paymentMethod`: Forma de pagamento (PIX, Cartão, Dinheiro, etc)
- `paymentDate`: Data em que o pagamento foi realizado

Backend - Procedimentos tRPC

Funções do Banco de Dados (server/db.ts)

createSale

```
export async function createSale(sale: InsertSale) {
  const db = await getDb();
  if (!db) {
    throw new Error("Database not available");
  }
  const result = await db.insert(sales).values(sale);
  return result;
}
```

Insere uma nova venda no banco de dados.

getSalesByUserId

```
export async function getSalesByUserId(userId: number) {
  const db = await getDb();
  if (!db) {
    throw new Error("Database not available");
  }
  const result = await db
    .select()
    .from(sales)
    .where(eq(sales.userId, userId))
    .orderBy(desc(sales.paymentDate));
  return result;
}
```

Retorna todas as vendas de um usuário, ordenadas por data (mais recentes primeiro).

updateSale

```
export async function updateSale(saleId: number, userId: number, updates:
Partial<Omit<InsertSale, 'userId'>>) {
  const db = await getDb();
  if (!db) {
    throw new Error("Database not available");
  }
  const result = await db
    .update(sales)
    .set(updates)
    .where(eq(sales.id, saleId));
  return result;
}
```

Atualiza os dados de uma venda existente.

deleteSale

```
export async function deleteSale(saleId: number, userId: number) {
  const db = await getDb();
  if (!db) {
    throw new Error("Database not available");
  }
  const result = await db
    .delete(sales)
    .where(eq(sales.id, saleId));
  return result;
}
```

Remove uma venda do banco de dados.

Procedimentos tRPC (server/routers.ts)

sales.create

```
create: protectedProcedure
  .input(z.object({
    productCode: z.string().min(1, "Código do produto é obrigatório"),
    clientName: z.string().min(1, "Nome do cliente é obrigatório"),
    value: z.string().transform(val =>
parseFloat(val)).pipe(z.number().positive("Valor deve ser positivo")),
    paymentMethod: z.string().min(1, "Forma de pagamento é obrigatória"),
    paymentDate: z.string().transform(val => new Date(val)),
  }))
  .mutation(async ({ ctx, input }) => {
    await createSale({
      userId: ctx.user.id,
      productCode: input.productCode,
      clientName: input.clientName,
      value: input.value.toString(),
      paymentMethod: input.paymentMethod,
      paymentDate: input.paymentDate,
    });
    return { success: true };
  }),
```

Cria uma nova venda. Requer autenticação.

sales.list

```
list: protectedProcedure
  .query(async ({ ctx }) => {
    const userSales = await getSalesByUserId(ctx.user.id);
    return userSales.map(sale => ({
      ...sale,
      value: typeof sale.value === 'string' ? parseFloat(sale.value) :
sale.value,
    }));
  }),
```

Retorna todas as vendas do usuário autenticado.

sales.stats

```
stats: protectedProcedure
  .query(async ({ ctx }) => {
    const userSales = await getSalesByUserId(ctx.user.id);

    const totalBruto = userSales.reduce((sum, sale) => {
      const value = typeof sale.value === 'string' ? parseFloat(sale.value)
: sale.value;
      return sum + value;
    }, 0);

    const totalLiquido = totalBruto * 0.3;
    const totalComissao = totalBruto * 0.7;

    const byPaymentMethod: Record<string, number> = {};
    userSales.forEach(sale => {
      const value = typeof sale.value === 'string' ? parseFloat(sale.value)
: sale.value;
      byPaymentMethod[sale.paymentMethod] =
(byPaymentMethod[sale.paymentMethod] || 0) + value;
    });

    return {
      totalBruto,
      totalLiquido,
      totalComissao,
      count: userSales.length,
      byPaymentMethod,
    };
  }),
```

Calcula estatísticas das vendas:

- **totalBruto**: Soma de todos os valores de venda
- **totalComissao**: 30% do valor bruto
- **count**: Quantidade de vendas
- **byPaymentMethod**: Total agrupado por forma de pagamento

sales.update

```
update: protectedProcedure
  .input(z.object({
    id: z.number(),
    productCode: z.string().min(1),
    clientName: z.string().min(1),
    value: z.string().transform(val =>
parseFloat(val)).pipe(z.number().positive()),
    paymentMethod: z.string().min(1),
    paymentDate: z.string().transform(val => new Date(val)),
  }))
  .mutation(async ({ ctx, input }) => {
    await updateSale(input.id, ctx.user.id, {
      productCode: input.productCode,
      clientName: input.clientName,
      value: input.value.toString(),
      paymentMethod: input.paymentMethod,
      paymentDate: input.paymentDate,
    });
    return { success: true };
  }),
```

Atualiza uma venda existente.

sales.delete

```
delete: protectedProcedure
  .input(z.object({
    id: z.number(),
  }))
  .mutation(async ({ ctx, input }) => {
    await deleteSale(input.id, ctx.user.id);
    return { success: true };
  }),
```

Deleta uma venda.

Frontend - Páginas e Componentes

Home.tsx - Página Inicial

Exibe o menu principal com três opções:

1. **Registrar Nova Venda** - Link para SalesForm
2. **Ver Relatórios** - Link para SalesReport
3. **Gerenciar Vendas** - Link para SalesManagement

```
export default function Home() {
  const { user, loading, error, isAuthenticated, logout } = useAuth();

  return (
    <div className="min-h-screen bg-background">
      {/* Header com nome do usuário e botão logout */}
      <header className="border-b bg-card">
        <div className="max-w-6xl mx-auto px-4 py-4 flex justify-between
items-center">
          <div className="flex items-center gap-2">
            <Gem className="w-8 h-8 text-primary" />
            <h1 className="text-2xl font-bold">{APP_TITLE}</h1>
          </div>
          <div className="flex items-center gap-4">
            <span className="text-sm text-muted-foreground">
              Bem-vindo, {user?.name || "Usuário"}!
            </span>
            <Button
              variant="outline"
              size="sm"
              onClick={logout}
            >
              Sair
            </Button>
          </div>
        </div>
      </header>

      {/* Conteúdo principal com cards de navegação */}
      <main className="flex-1 space-y-6 p-4">
        {/* Cards para cada seção */}
      </main>
    </div>
  );
}
```

SalesForm.tsx - Formulário de Registro

Permite registrar uma nova venda com os campos:

- Código do Produto
- Nome do Cliente

- Valor
- Forma de Pagamento
- Data do Pagamento

```tsx
export default function SalesForm() {
  const [formData, setFormData] = useState({
    productCode: "",
    clientName: "",
    value: "",
    paymentMethod: "PIX",
    paymentDate: format(new Date(), "yyyy-MM-dd"),
  });

  const createMutation = trpc.sales.create.useMutation({
    onSuccess: () => {
      // Limpar formulário
      setFormData({...});
      toast.success("Venda registrada com sucesso!");
    },
    onError: (error) => {
      toast.error(error.message);
    },
  });

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    createMutation.mutate(formData);
  };

  return (
    <div className="w-full min-h-screen flex flex-col">
      {/* Header com botão voltar */}
      <header className="border-b bg-card">
        {/* ... */}
      </header>

      {/* Formulário */}
      <main className="flex-1 space-y-6 p-4">
        <Card>
          <CardHeader>
            <CardTitle>Registrar Nova Venda</CardTitle>
          </CardHeader>
          <CardContent>
            <form onSubmit={handleSubmit} className="space-y-4">
              {/* Campos do formulário */}
            </form>
          </CardContent>
        </Card>
      </main>
```

```
    </div>
  );
}
```

SalesReport.tsx - Relatórios com Filtros

Exibe relatórios de vendas com filtros avançados e exportação em Excel.

Filtros Implementados:

1. **Cliente** - Busca por nome do cliente (case-insensitive)
2. **Forma de Pagamento** - Dropdown com todas as formas registradas
3. **Data Inicial** - Filtro por data de início
4. **Data Final** - Filtro por data de término

```
const [filterPaymentMethod, setFilterPaymentMethod] = useState<string>("");
const [filterClientName, setFilterClientName] = useState<string>("");
const [filterStartDate, setFilterStartDate] = useState<string>("");
const [filterEndDate, setFilterEndDate] = useState<string>("");

const filteredSales = useMemo(() => {
  if (!sales) return [];

  return sales.filter(sale => {
    if (filterPaymentMethod && sale.paymentMethod !== filterPaymentMethod)
return false;
    if (filterClientName &&
!sale.clientName.toLowerCase().includes(filterClientName.toLowerCase()))
return false;

    if (filterStartDate) {
      const saleDate = new Date(sale.paymentDate);
      const startDate = new Date(filterStartDate);
      if (saleDate < startDate) return false;
    }

    if (filterEndDate) {
      const saleDate = new Date(sale.paymentDate);
      const endDate = new Date(filterEndDate);
      endDate.setHours(23, 59, 59, 999);
      if (saleDate > endDate) return false;
    }

    return true;
  });
}, [sales, filterPaymentMethod, filterClientName, filterStartDate,
filterEndDate]);

const filteredStats = useMemo(() => {
  const totalBruto = filteredSales.reduce((sum, sale) => sum + sale.value,
0);
  const totalComissao = totalBruto * 0.3;

  return {
    totalBruto,
    totalComissao,
    count: filteredSales.length,
  };
}, [filteredSales]);
```

Exportação em Excel:

```
const exportToExcel = () => {
  if (!sales || sales.length === 0) {
    alert("Nenhuma venda para exportar");
    return;
  }

  const dataToExport = sales.map((sale) => ({
    "Código do Produto": sale.productCode,
    "Cliente": sale.clientName,
    "Valor": sale.value,
    "Forma de Pagamento": sale.paymentMethod,
    "Data do Pagamento": formatDate(sale.paymentDate),
  }));

  const summaryData = [
    { "Métrica": "Total Bruto", "Valor": stats?.totalBruto || 0 },
    { "Métrica": "Comissão (30%)", "Valor": stats?.totalComissao || 0 },
    { "Métrica": "Quantidade de Vendas", "Valor": stats?.count || 0 },
  ];

  const workbook = XLSX.utils.book_new();
  const vendas = XLSX.utils.json_to_sheet(dataToExport);
  const resumo = XLSX.utils.json_to_sheet(summaryData);

  XLSX.utils.book_append_sheet(workbook, vendas, "Vendas");
  XLSX.utils.book_append_sheet(workbook, resumo, "Resumo");

  const fileName = `relatorio_vendas_${format(new Date(), "dd-MM-
yyyy")}.xlsx`;
  XLSX.writeFile(workbook, fileName);
};
```

SalesManagement.tsx - Gerenciamento de Vendas

Permite editar e deletar vendas com confirmação.

Funcionalidades:

- Tabela com todas as vendas do usuário
- Botão “Editar” abre modal com formulário preenchido

- Botão “Deletar” abre modal de confirmação
- Sincronização automática após editar/deletar

```
const [editingSale, setEditingSale] = useState<Sale | null>(null);
const [deletingSaleId, setDeletingSaleId] = useState<number | null>(null);

const updateMutation = trpc.sales.update.useMutation({
  onSuccess: () => {
    setEditingSale(null);
    utils.sales.list.invalidate();
    toast.success("Venda atualizada com sucesso!");
  },
});

const deleteMutation = trpc.sales.delete.useMutation({
  onSuccess: () => {
    setDeletingSaleId(null);
    utils.sales.list.invalidate();
    toast.success("Venda deletada com sucesso!");
  },
});
```

Funcionalidades Implementadas

✅ Funcionalidades Principais

1. **Autenticação**
   - Login via Manus OAuth
   - Logout seguro
   - Isolamento de dados por usuário
2. **Registro de Vendas**
   - Formulário com validação
   - Campos: código, cliente, valor, forma de pagamento, data
   - Feedback visual (toasts)

- Limpeza automática após envio

3. **Visualização de Relatórios**
   - Total Bruto (soma de todas as vendas)
   - Comissão (30% do bruto)
   - Quantidade de vendas
   - Tabela com detalhes de cada venda

4. **Filtros Avançados**
   - Busca por cliente (em tempo real)
   - Filtro por forma de pagamento
   - Filtro por período de datas
   - Botão "Limpar Filtros"
   - Estatísticas refletem dados filtrados

5. **Exportação em Excel**
   - Botão "Exportar Excel" na página de relatórios
   - Arquivo com 2 abas: "Vendas" e "Resumo"
   - Nome automático com data (relatorio_vendas_DD-MM-YYYY.xlsx)

6. **Gerenciamento de Vendas**
   - Editar venda (modal com formulário preenchido)
   - Deletar venda (com confirmação)
   - Sincronização automática

7. **Navegação**
   - Botão "Página Inicial" em todas as páginas
   - Menu principal com 3 opções
   - Responsive design (mobile, tablet, desktop)

Guia de Uso

1. Acessar o Sistema

1. Acesse a URL do sistema
2. Faça login com sua conta Manus
3. Você será redirecionado para a página inicial

2. Registrar uma Venda

1. Clique em "Registrar Nova Venda"
2. Preencha os campos:
   - **Código do Produto**: Ex: "JOI001"
   - **Cliente**: Nome do cliente
   - **Valor**: Valor em reais (ex: 150.50)
   - **Forma de Pagamento**: Selecione PIX, Cartão, Dinheiro, etc
   - **Data do Pagamento**: Data em que o pagamento foi realizado
3. Clique em "Registrar Venda"
4. Você verá uma mensagem de sucesso

3. Visualizar Relatórios

1. Clique em "Ver Relatórios"
2. Você verá:
   - **Filtros** na parte superior
   - **Estatísticas** (Total Bruto, Comissão 30%, Quantidade)
   - **Tabela** com todas as vendas
   - **Botão Exportar Excel** para baixar dados

4. Usar Filtros

1. Na página de relatórios, preencha os filtros desejados:
   - **Cliente**: Digite o nome (busca parcial)

- **Forma de Pagamento**: Selecione no dropdown
- **Data Inicial**: Selecione a data de início
- **Data Final**: Selecione a data de término

2. Os dados serão filtrados automaticamente
3. Clique em "Limpar Filtros" para resetar

5. Exportar Relatório em Excel

1. Na página de relatórios, clique em "Exportar Excel"
2. Um arquivo será baixado com o nome: `relatorio_vendas_DD-MM-YYYY.xlsx`
3. O arquivo contém 2 abas:
   - **Vendas**: Detalhes de cada venda
   - **Resumo**: Totais e estatísticas

6. Editar uma Venda

1. Clique em "Gerenciar Vendas"
2. Localize a venda na tabela
3. Clique em "Editar"
4. Modifique os campos desejados
5. Clique em "Salvar Alterações"

7. Deletar uma Venda

1. Clique em "Gerenciar Vendas"
2. Localize a venda na tabela
3. Clique em "Deletar"
4. Confirme a exclusão no modal
5. A venda será removida do banco de dados

Cálculos Implementados

Total Bruto

```
Total Bruto = Σ (valor de cada venda)
```

Comissão (30%)

```
Comissão = Total Bruto × 0.30
```

Exemplo

- Venda 1: R$ 100,00
- Venda 2: R$ 200,00
- Venda 3: R$ 150,00

Total Bruto: R$450,00 * *Comissão*(30 135,00

Tecnologias Utilizadas

Tecnologia	Versão	Uso
React	19	Framework frontend
TypeScript	-	Tipagem estática
Tailwind CSS	4	Estilos
tRPC	11	RPC type-safe
Express.js	4	Servidor backend
Drizzle ORM	-	ORM para banco de dados
MySQL	-	Banco de dados
XLSX	0.18.5	Exportação Excel
date-fns	-	Formatação de datas
Zod	-	Validação de schemas

Segurança

- ✅ Autenticação obrigatória via Manus OAuth
- ✅ Isolamento de dados por usuário (userId)
- ✅ Validação de entrada com Zod
- ✅ Proteção de rotas com `protectedProcedure`
- ✅ Senhas não armazenadas (OAuth)
- ✅ HTTPS em produção

Próximas Melhorias Sugeridas

1. **Gráficos visuais** - Adicionar gráficos de barras/pizza para visualizar dados

2. **Agendamento de relatórios** - Enviar relatórios por email automaticamente
3. **Múltiplos usuários** - Suporte para equipes/colaboradores
4. **Backup automático** - Sistema de backup de dados
5. **API pública** - Integração com sistemas externos
6. **Notificações** - Alertas de vendas importantes
7. **Dashboard** - Visão geral com KPIs principais

Desenvolvido com ❤️ usando Manus