🔧 Correção do Erro de Servidor no Chat do OrganiZen

Data: 13/11/2024
Erro Original: “Application error: a server-side exception has occurred (Digest: 4256790555)”

📋 Problema Identificado

O erro ocorria quando o usuário acessava a página de chat no preview do OrganiZen. A causa raiz foi identificada como um **problema de serialização de dados** entre componentes Server-Side e Client-Side no Next.js 14.

Causa Técnica

O Next.js App Router (v14) tem limitações estritas sobre quais tipos de dados podem ser passados de Server Components para Client Components através de props. Especificamente:

- **Enums do Prisma** não podem ser serializados corretamente
- O campo `role` no modelo `User` é definido como um `UserRole` enum
- Quando passado diretamente como prop, causa erro de serialização

---

✅ Solução Implementada

1. Correção na Página de Chat (`app/chat/page.tsx`)

Antes:

```
const users = await prisma.user.findMany({
  where: {
    companyId,
    id: { not: userId }
  },
  select: {
    id: true,
    name: true,
    email: true,
    role: true,  // ❌ Enum não serializado
    image: true
  },
  orderBy: { name: 'asc' }
});

return (
  <ChatContent
    users={users}  // ❌ Passando enum diretamente
    // ...
  />
);
```

Depois:

```
const usersData = await prisma.user.findMany({
  where: {
    companyId,
    id: { not: userId }
  },
  select: {
    id: true,
    name: true,
    email: true,
    role: true,
    image: true
  },
  orderBy: { name: 'asc' }
});

// ✅ Converter enum para string para serialização adequada
const users = usersData.map(user => ({
  id: user.id,
  name: user.name,
  email: user.email,
  role: user.role as string,  // ✅ Conversão explícita
  image: user.image
}));

return (
  <ChatContent
    users={users}  // ✅ Dados serializáveis
    // ...
  />
);
```

---

2. Correção na API de Conversas (`app/api/chat/conversations/route.ts`)

Antes:

```
const users = await prisma.user.findMany({
  where: {
    id: { in: userIds }
  },
  select: {
    id: true,
    name: true,
    email: true,
    image: true,
    role: true  // ❌ Enum não convertido
  }
});
```

Depois:

```
const usersData = await prisma.user.findMany({
  where: {
    id: { in: userIds }
  },
  select: {
    id: true,
    name: true,
    email: true,
    image: true,
    role: true
  }
});

// ✅ Converter enum para string para serialização JSON
const users = usersData.map(user => ({
  id: user.id,
  name: user.name,
  email: user.email,
  image: user.image,
  role: user.role as string  // ✅ Conversão explícita
}));
```

🎯 Arquivos Modificados

Páginas (Server Components)

1. ✅ `/app/chat/page.tsx` - Página de chat em tempo real
2. ✅ `/app/messages/page.tsx` - Sistema de mensagens internas
3. ✅ `/app/tasks/page.tsx` - Gerenciamento de tarefas
4. ✅ `/app/shifts/page.tsx` - Gerenciamento de turnos

APIs (Route Handlers)

1. ✅ `/app/api/chat/conversations/route.ts` - API de conversas de chat

📝 Resumo das Correções por Arquivo

1. app/chat/page.tsx

- ✅ Converteu enum `UserRole` para string na lista de usuários
- ✅ Adicionou mapeamento explícito antes de passar dados para componente client

2. app/messages/page.tsx

- ✅ Converteu enum `UserRole` em `sender` e `receiver` de todas as mensagens
- ✅ Converteu enum na lista de usuários disponíveis
- ✅ Manteve conversão de datas (já existente)

3. app/tasks/page.tsx

- ✅ Converteu enum `TaskStatus` para string
- ✅ Converteu enum `UserRole` no objeto `user` de cada tarefa
- ✅ Converteu enum na lista de usuários disponíveis

4. app/shifts/page.tsx

- ✅ Converteu enum `UserRole` no objeto `user` de cada turno
- ✅ Converteu enum na lista de usuários disponíveis
- ✅ Manteve conversão de datas (já existente)

5. app/api/chat/conversations/route.ts

- ✅ Converteu enum `UserRole` antes de retornar JSON
- ✅ Garantiu serialização correta em respostas de API

🧪 Testes Realizados

✅ Servidor Local

- Servidor Next.js iniciado sem erros
- Build bem-sucedido
- Nenhum erro de compilação TypeScript

⏳ Próximos Passos para Teste Completo

1. Acessar a página de chat no preview do Abacus.AI
2. Verificar se a lista de usuários carrega corretamente
3. Testar envio e recebimento de mensagens
4. Verificar status online/offline dos usuários

📚 Lições Aprendidas

Regra Geral para Next.js 14 App Router:

Ao passar dados de Server Components para Client Components, **SEMPRE**:

1. ✅ Converter enums para strings
2. ✅ Converter Dates para ISO strings
3. ✅ Evitar passar instâncias de classes
4. ✅ Manter objetos simples (Plain Old JavaScript Objects - POJOs)

Padrão Recomendado:

```
// ❌ EVITAR
return <ClientComponent data={rawDatabaseData} />

// ✅ RECOMENDADO
const serializedData = rawDatabaseData.map(item => ({
  ...item,
  enumField: item.enumField as string,
  dateField: item.dateField.toISOString(),
}));
return <ClientComponent data={serializedData} />
```

🚀 Status

- ✅ **Todas as correções implementadas (5 arquivos)**
- ✅ **Servidor local funcionando sem erros**
- ✅ **Build bem-sucedido**
- ⏳ **Aguardando teste no preview/produção**

💡 Impacto das Correções

Páginas Corrigidas

- ✅ Chat em tempo real
- ✅ Sistema de mensagens internas
- ✅ Gerenciamento de tarefas
- ✅ Gerenciamento de turnos

Benefícios

- 🛡️ Elimina erros de serialização em todas as páginas principais
- 🚀 Melhora a estabilidade do aplicativo
- 📊 Garante que dados sejam passados corretamente entre server e client
- 🔧 Previne futuros erros similares

📝 Notas Adicionais

Esta correção é uma **best practice** e deve ser aplicada em todos os lugares onde:
- Dados do banco (com enums ou Dates) são passados de Server para Client Components
- APIs retornam dados com enums (embora NextResponse.json geralmente lide bem, é melhor ser explícito)

Manutenção Futura: Sempre verificar este padrão ao criar novas páginas com Server Components que passam dados para Client Components.