



Guia Frontend - Cronograma UBSF



Correções Implementadas no Backend



Problemas Resolvidos:

1. **CORS corrigido** - Frontend pode acessar API diretamente (sem cors-anywhere)
 2. **Autenticação implementada** - Cronogramas agora filtrados por usuário
 3. **Segurança aprimorada** - Cada usuário vê apenas seus cronogramas
-



Implementações Necessárias no Frontend



1. Remover cors-anywhere e usar API direta

✗ Remover:

```
// Código antigo problemático
const API_URL = 'https://cors-anywhere.herokuapp.com/https://erika-ubsf.vercel.app';
```



Usar:

```
// Código novo funcionando
const API_URL = 'https://erika-ubsf.vercel.app';
```



2. Implementar Feedback de Cadastro

Problema atual: Usuário é redirecionado sem saber se cadastro funcionou.

Solução - Exemplo React Native:

```
// Exemplo de tela de cadastro com feedback
const [loading, setLoading] = useState(false);
const [feedback, setFeedback] = useState({ type: '', message: '' });

const handleCadastro = async (userData) => {
  setLoading(true);
  setFeedback({ type: '', message: '' });

  try {
    const response = await fetch('https://erika-ubsf.vercel.app/api/auth/cadastro', {
      method: 'POST',
```

```

    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(userData),
  });

  const data = await response.json();

  if (data.success) {
    // ✅ Sucesso
    setFeedback({
      type: 'success',
      message: '✅ Cadastro realizado com sucesso! Redirecionando...'
    });

    // Salvar token se necessário
    await AsyncStorage.setItem('userToken', data.data.token);

    // Redirecionar após mostrar feedback
    setTimeout(() => {
      navigation.navigate('Login'); // ou diretamente para home
    }, 2000);
  } else {
    // ❌ Erro retornado pela API
    setFeedback({
      type: 'error',
      message: `❌ ${data.message}`
    });
  }
} catch (error) {
  // ❌ Erro de conexão
  setFeedback({
    type: 'error',
    message: '❌ Erro de conexão. Verifique sua internet.'
  });
} finally {
  setLoading(false);
}
};

// No JSX:
{feedback.message && (
  <View style={styles.feedback,
    feedback.type === 'success' ? styles.success : styles.error}>
    <Text>{feedback.message}</Text>
  </View>
)}

{loading && <ActivityIndicator size="large" color="#0000ff" />}

```

3. 🔑 Implementar Autenticação nos Cronogramas

IMPORTANTE: Todos os endpoints de cronogramas agora requerem autenticação!

Exemplo de serviço API atualizado:

```
// services/api.js
import AsyncStorage from '@react-native-async-storage/async-storage';

const API_BASE_URL = 'https://erika-ubsf.vercel.app';

const getAuthHeaders = async () => {
  const token = await AsyncStorage.getItem('userToken');
  return {
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${token}`
  };
};

export const api = {
  // Autenticação
  async cadastrar(userData) {
    const response = await fetch(`${API_BASE_URL}/api/auth/cadastro`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(userData),
    });
    return response.json();
  },

  async login(credentials) {
    const response = await fetch(`${API_BASE_URL}/api/auth/login`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(credentials),
    });
    return response.json();
  },

  // Cronogramas (AGORA COM AUTENTICAÇÃO OBRIGATÓRIA)
  async getCronogramas(params = {}) {
    const queryString = new URLSearchParams(params).toString();
    const headers = await getAuthHeaders();

    const response = await fetch(`${API_BASE_URL}/api/cronogramas?${queryString}`, {
      headers
    });
    return response.json();
  },
}
```

```

async createCronograma(data) {
  const headers = await getAuthHeaders();

  const response = await fetch(`${API_BASE_URL}/api/cronogramas`, {
    method: 'POST',
    headers,
    body: JSON.stringify(data),
  });
  return response.json();
},

async getCronograma(id) {
  const headers = await getAuthHeaders();

  const response = await fetch(`${API_BASE_URL}/api/cronogramas/${id}`, {
    headers
  });
  return response.json();
},

// ... outros métodos também precisam dos headers de auth
};

```

4. Gerenciar Estado de Autenticação

Exemplo de Context de Autenticação:

```

// contexts/AuthContext.js
import React, { createContext, useState, useContext, useEffect } from 'react';
import AsyncStorage from '@react-native-async-storage/async-storage';

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [token, setToken] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    loadStoredAuth();
  }, []);

  const loadStoredAuth = async () => {
    try {
      const storedToken = await AsyncStorage.getItem('userToken');
      const storedUser = await AsyncStorage.getItem('userData');

      if (storedToken && storedUser) {
        setToken(storedToken);
      }
    }
  }
}

```

```

        setUser(JSON.parse(storedUser));
    }
    } catch (error) {
        console.log('Erro ao carregar auth:', error);
    } finally {
        setLoading(false);
    }
};

const login = async (credentials) => {
    try {
        const response = await api.login(credentials);

        if (response.success) {
            const { token: newToken, usuario } = response.data;

            await AsyncStorage.setItem('userToken', newToken);
            await AsyncStorage.setItem('userData', JSON.stringify(usuario));

            setToken(newToken);
            setUser(usuario);

            return { success: true };
        } else {
            return { success: false, message: response.message };
        }
    } catch (error) {
        return { success: false, message: 'Erro de conexão' };
    }
};

const logout = async () => {
    await AsyncStorage.removeItem('userToken');
    await AsyncStorage.removeItem('userData');
    setToken(null);
    setUser(null);
};

return (
    <AuthContext.Provider value={{
        user,
        token,
        loading,
        login,
        logout,
        isAuthenticated: !!token
    }}>
        {children}
    </AuthContext.Provider>
);
};

export const useAuth = () => {

```

```

const context = useContext(AuthContext);
if (!context) {
  throw new Error('useAuth deve ser usado dentro de AuthProvider');
}
return context;
};

```

5. Tratar Erros de Autenticação

Interceptar respostas 401 (Não autorizado):

```

// utils/apiInterceptor.js
export const handleApiResponse = async (response) => {
  const data = await response.json();

  if (response.status === 401) {
    // Token inválido ou expirado
    await AsyncStorage.removeItem('userToken');
    await AsyncStorage.removeItem('userData');

    // Redirecionar para login
    navigation.navigate('Login');

    throw new Error('Sessão expirada. Faça login novamente.');
```

6. Exemplo de Tela de Cronogramas Atualizada

```

import React, { useState, useEffect } from 'react';
import { View, Text, FlatList, Alert } from 'react-native';
import { useAuth } from '../contexts/AuthContext';
import { api } from '../services/api';

export default function CronogramasList() {
  const [cronogramas, setCronogramas] = useState([]);
  const [loading, setLoading] = useState(true);
  const { user, isAuthenticated } = useAuth();

  useEffect(() => {
    if (isAuthenticated) {
      loadCronogramas();
    }
  }, [isAuthenticated]);

  const loadCronogramas = async () => {

```

```

try {
  setLoading(true);
  const response = await api.getCronogramas();

  if (response.success) {
    setCronogramas(response.data.cronogramas);
  } else {
    Alert.alert('Erro', response.message);
  }
} catch (error) {
  Alert.alert('Erro', 'Não foi possível carregar cronogramas');
} finally {
  setLoading(false);
}
};

if (!isAuthenticated) {
  return (
    <View style={styles.container}>
      <Text>Você precisa fazer login para ver os cronogramas</Text>
    </View>
  );
}

return (
  <View style={styles.container}>
    <Text style={styles.welcome}>
      Olá, {user?.nome}! Seus cronogramas:
    </Text>

    {cronogramas.length === 0 ? (
      <Text style={styles.empty}>
        Você ainda não possui cronogramas. Crie seu primeiro!
      </Text>
    ) : (
      <FlatList
        data={cronogramas}
        renderItem={({ item }) => (
          <View style={styles.cronogramaItem}>
            <Text>{item.nomeUBSF} - {item.mes}/{item.ano}</Text>
            <Text>Atividades: {item.atividades.length}</Text>
          </View>
        )}
        keyExtractor={(item) => item.id}
        refreshing={loading}
        onRefresh={loadCronogramas}
      />
    )}
  </View>
);
}

```


Checklist de Implementação

- ☐ Remover cors-anywhere do código
- ☐ Atualizar URL da API para acesso direto
- ☐ Implementar feedback visual no cadastro
- ☐ Adicionar headers de autenticação em todas as requisições de cronograma
- ☐ Implementar gerenciamento de estado de autenticação
- ☐ Tratar erros 401 (sessão expirada)
- ☐ Testar fluxo completo: cadastro → login → cronogramas

Resultado Esperado

Após essas implementações:

- ☒ Usuário saberá se cadastro deu certo ou não
- ☒ Cada usuário verá apenas seus próprios cronogramas
- ☒ Sistema de autenticação funcionando corretamente
- ☒ Experiência do usuário muito melhor

 **Dica:** Teste sempre o fluxo completo: Cadastro → Login → Ver cronogramas → Criar cronograma novo