Logins:

Administrador: admin@wayne.com

Senha: 123456

Gerente de Transporte: transportemanager@wayne.com
Senha: 147258

Gerente de Segurança: segurancamanager@wayne.com
Senha: 987654

Gerente de TI: timanager@wayne.com
Senha: 741852

Colaborador de Transporte: transporteemployee@wayne.com
Senha: 852963

Colaborador de Segurança: segurancaemployee@wayne.com
Senha: 258369

Colaborador de TI: tiemployee@wayne.com
Senha: 753951

Documentação do Projeto - Indústrias Wayne

1. Visão Geral do Projeto

Este projeto é um painel de gerenciamento para as Indústrias Wayne. Ele permite que os usuários visualizem e gerenciem os recursos da empresa, com diferentes níveis de acesso baseados no cargo do usuário.

Tecnologias Utilizadas

   **Frontend: HTML, CSS, JavaScript
   **Framework CSS: Bootstrap 5
   **Backend & Banco de Dados: Firebase (Authentication e Firestore)
   **Bibliotecas JavaScript:
      Chart.js (para visualização de dados no dashboard)
     VLibras (plugin de acessibilidade)

2. Estrutura do Projeto

A estrutura de arquivos do projeto é organizada da seguinte forma:

```
├────firebase.config.js       # Configuração do Firebase
```

```
├──── index.html                # Página principal da aplicação (dashboard)
├──── login.html                # Página de login
├──── PROJECT_DOCUMENTATION.md # Documentação do projeto
├──── css/
│   └──── style.css             # Estilos personalizados
├──── images/                   # Imagens e favicons
└──── js/
    ├──── app.js                # Lógica principal da aplicação e event listeners
    ├──── auth.js               # Funções de autenticação (login, logout)
    ├──── database.js           # Funções de interação com o Firestore
    ├──── resources.js          # (Arquivo com lógica de recursos, aparentemente não
utilizado)
    └──── ui.js                 # Funções para manipulação e renderização da interface
do usuário
```

3. Banco de Dados (Firestore)

O Firestore é utilizado como banco de dados NoSQL para armazenar os dados da aplicação.

Coleções

Existem duas coleções principais no banco de dados:

1. `users`: Armazena as informações de função dos usuários.
   Documento: O ID do documento é o UID do usuário do Firebase Authentication.
   Campos:
     `role` (string): A função do usuário (`admin`, `manager`, ou `employee`).

2. **`resources`**: Armazena os recursos da empresa.
   Documento: ID gerado automaticamente pelo Firestore.
   Campos:
     `name` (string): Nome do recurso.
     `type` (string): Tipo do recurso (ex: "Equipamento", "Veículo").
     `status` (string): Status do recurso (ex: "Operacional", "Em Manutenção", "Inativo").
     `createdAt` (timestamp): Data e hora da criação do registro.

Regras de Segurança

As regras de segurança do Firestore não foram fornecidas. É crucial definir regras de segurança para proteger os dados contra acesso não autorizado. As regras devem garantir que:

- Apenas usuários autenticados possam ler e escrever dados.
- Apenas usuários com a função `admin` ou `manager` possam criar, editar e excluir recursos.
- Todos os usuários autenticados possam ler os recursos (para o dashboard).
- Usuários só possam ler seu próprio documento na coleção `users`.

   Apenas `admins` possam criar ou modificar documentos na coleção `users` (para definir as funções).

Regras aplicadas:

```
rules_version = '2';
service cloud.firestore {
 match /databases/{database}/documents {
  match /users/{userId} {
   allow read: if request.auth.uid == userId;
   allow write: if get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'admin';
  }

  match /resources/{resourceId} {
   allow read: if request.auth != null;
   allow create, update, delete: if get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role in ['admin', 'manager'];
  }
 }
}
```

4. Gerenciamento de Usuários

Níveis de Acesso

O sistema possui três níveis de acesso:

1. **`admin`**: Acesso total. Pode ver o dashboard e gerenciar (adicionar, editar, apagar) todos os recursos. O nome de exibição é "admin".
2. **`manager`**: Acesso de gerente. Pode ver o dashboard e gerenciar (adicionar, editar, apagar) todos os recursos. O nome de exibição é "Gerente".
3. **`employee`**: Acesso de colaborador. Pode apenas visualizar o dashboard. Não tem acesso à página de gerenciamento de recursos. O nome de exibição é "Colaborador".

Como Adicionar Novos Usuários e Definir Níveis de Acesso

O processo é feito em duas etapas:

1. **Criar o Usuário no Firebase Authentication:**
   * Vá para o seu projeto no [console do Firebase](https://console.firebase.google.com/).
   * Navegue até a seção **Authentication**.
   * Clique em **"Add user"** e preencha o email e a senha do novo usuário.
   * Após criar o usuário, copie o **User UID**.

2. **Definir o Nível de Acesso no Firestore:**
   * Navegue até a seção **Firestore Database** no console do Firebase.

  *   Selecione a coleção **`users`**.
  *   Clique em **"Add document"**.
  *   Cole o **User UID** que você copiou no campo **Document ID**.
  *   Adicione um campo (Add field) com o nome **`role`**.
  *   Defina o valor do campo `role` para `admin`, `manager` ou `employee`, de acordo com o nível de acesso desejado.
  *   Clique em **"Save"**.

5. Funcionalidades

*   **Autenticação de Usuários:** Login e logout seguros utilizando o Firebase Authentication.
*   **Dashboard:** Exibe um resumo dos recursos, incluindo:
  *   Número total de recursos.
  *   Contagem de recursos por status (Operacional, Em Manutenção, Inativo).
  *   Gráficos de pizza/rosca para visualizar a distribuição de recursos por tipo e por status.
*   **Gerenciamento de Recursos (Apenas para Admin/Manager):**
  *   Listagem de todos os recursos em uma tabela.
  *   Exibição do ID, nome, tipo e status de cada recurso.
  *   Adicionar novos recursos através de um modal.
  *   Editar informações de recursos existentes.
  *   Excluir recursos.

Como Adicionar Novos Tipos de Recursos (Ex: Móveis)

O banco de dados Firestore não possui um esquema rígido, o que significa que você não precisa "pré-definir" os tipos de recursos. Você pode simplesmente começar a salvar recursos com o novo tipo.

Opção 1: Adicionar Através da Aplicação (Recomendado)

Depois que a opção "Móveis" foi adicionada ao formulário na interface, o processo é simples:

1. Faça login na aplicação com uma conta `admin` ou `manager`.
2. Vá para a página de **Recursos**.
3. Clique no botão **"Adicionar Recurso"**.
4. Preencha o nome e selecione **"Móveis"** no campo **Tipo**.
5. Selecione um status e clique em **"Salvar"**.

Isso criará um novo documento na coleção `resources` com o campo `type` definido como "Móveis".

Opção 2: Adicionar Manualmente no Console do Firebase

Você também pode adicionar um novo recurso diretamente no banco de dados do Firestore.

1. Vá para o seu projeto no [console do Firebase](https://console.firebase.google.com/).
2. Navegue até a seção **Firestore Database**.
3. Selecione a coleção **`resources`**.
4. Clique em **"Add document"**.
5. O Firestore irá gerar um ID de documento para você.
6. Adicione os seguintes campos:
   * **`name`** (string): O nome do móvel (ex: "Cadeira de Escritório Ergonômica").
   * **`type`** (string): Defina o valor como **"Móveis"**.
   * **`status`** (string): Defina o status (ex: "Operacional").
   * **`createdAt`** (timestamp): Adicione um timestamp para a data de criação.
7. Clique em **"Save"**.

#style.css

1. Tema Escuro: Todas as cores são escolhidas para criar uma interface escura

2. Hierarquia Visual: Diferentes tons de escuro (#121212, #1e1e1e, #2a2a2a) criam profundidade

3. Feedback Visual: Estados de foco e hover fornecem feedback interativo

4. Contraste: Cores claras sobre fundos escuros garantem boa legibilidade

5. Consistência: Padrão visual uniforme em todos os componentes

Este código implementa um tema escuro moderno compatível com Bootstrap, mantendo acessibilidade e usabilidade.

#app.js

1. Fluxo de Autenticação: Monitora estado de login e configura a interface conforme o papel do usuário

2. Sistema de Navegação: Roteamento client-side com controle de acesso baseado em papéis

3. Gestão de Estado: Mantém o papel do usuário e referências de UI globalmente

4. Delegação de Eventos: Gerencia elementos dinâmicos eficientemente

5. CRUD de Recursos: Operações completas de Create, Read, Update e Delete

6. Modais Interativos: Formulários reutilizáveis para adicionar/editar recursos

O código segue princípios de separação de preocupações e fornece uma base sólida para uma aplicação web single-page com autenticação e autorização.

#auth.js

1. Persistência de Sessão
SESSION: Login válido apenas durante a sessão do navegador

Segurança: Impede que usuários permaneçam logados em dispositivos compartilhados

Experiência: Mantém o login durante a navegação na aplicação

2. Fluxo de Login
Validação: Impede envio tradicional do formulário

Autenticação: Integração com Firebase Auth

Feedback: Mensagens de erro contextuais para o usuário

Redirecionamento: Automático para a aplicação após login

3. Fluxo de Logout
Limpeza de Sessão: Remove credenciais do Firebase

Redirecionamento: Força retorno à página de login

Tratamento de Erros: Log de possíveis falhas

4. Controle de Acesso
Proteção de Rotas: Impede acesso não autenticado à aplicação

Redirecionamento Inteligente: Baseado no estado atual do usuário

Experiência Coerente: Usuários sempre são direcionados para o local apropriado

Casos de Uso Cobertos:
✅ Primeiro acesso (→ login)

✅ Login bem-sucedido (→ aplicação)

✅ Tentativa de acesso sem autenticação (→ login)

✅ Logout manual (→ login)

✅ Sessão expirada (→ login)

✅ Acesso à página de login já autenticado (→ aplicação)

Este sistema garante que usuários não autenticados não consigam acessar áreas restritas da aplicação, mantendo a segurança enquanto proporciona uma experiência de usuário fluida.

#database.js

1. Estrutura de Dados
Coleção 'resources': Armazena todos os recursos do sistema

Coleção 'users': Armazena informações adicionais dos usuários (papéis)

Documentos: Cada recurso/usuário é um documento com campos específicos

2. Operações CRUD Completas
Create: saveResource() sem ID + campo createdAt

Read: getResources() (lista) e getResource() (individual)

Update: saveResource() com ID existente

Delete: deleteResource() por ID

3. Boas Práticas Implementadas
Ordenação: Recursos ordenados por createdAt descendente

Timestamps: Uso de serverTimestamp() para consistência

Validação: Verificação de existência de documentos

Tratamento de Erros: Fallbacks seguros para funções críticas

4. Sistema de Autorização
Papéis de Usuário: employee, manager, admin

Consulta Segura: Busca otimizada por UID do usuário

Fallback Robusto: Sempre retorna 'employee' em caso de problemas

Fluxo de Dados Típico:

Firebase Auth (UID) → Firestore Users (Role) → Controle de Acesso → Operações em Resources

Campos dos Recursos:
name: Nome identificador do recurso

type: Categoria/classificação do recurso

status: Estado atual do recurso

createdAt: Data/hora de criação (automática)

id: Identificador único do documento (automático)

Este código fornece uma base sólida para operações de banco de dados seguras e eficientes, com controle de acesso baseado em papéis e tratamento robusto de erros.

#resources.js

1. Estrutura de Dados do Recurso
Campos Obrigatórios: name, type, quantity, location, notes

Metadados Automáticos: createdAt (timestamp), createdBy (ID do usuário)

Validação: Trim de strings e conversão numérica

2. Fluxo da Interface

Botão "Adicionar" → Abre Modal → Preenche Formulário → Submit →
Valida Dados → Adiciona ao Firestore → Fecha Modal → Limpa Formulário

3. Tratamento de Estados
Sucesso: Fecha modal, limpa formulário, log no console

Erro: Mantém modal aberto, exibe alerta, log de erro

Carregamento: Feedback através do comportamento do formulário

4. Integrações
Firestore: Conexão através da coleção resources

Firebase Auth: Registro do usuário criador

Bootstrap: Modais para interface amigável

Window Object: Disponibilização global da função

Boas Práticas Implementadas:
✅ Validação e sanitização de dados

✅ Tratamento robusto de erros

✅ Metadados automáticos (timestamp e usuário)

✅ Interface responsiva com modais

✅ Separação de preocupações (UI vs. Dados)

✅ Função exposta para reutilização

Este código fornece uma base sólida para operações de criação de recursos com interface amigável e tratamento adequado de todos os cenários possíveis.

#ui.js

1. Controle de Acesso Baseado em Papéis
Admin/Manager: Acesso completo à gestão de recursos

Employee: Apenas visualização do dashboard

Interface Adaptativa: Elementos mostrados/ocultos conforme permissões

2. Dashboard com Visualizações
Cards de Resumo: Métricas principais em tempo real

Gráficos Interativos: Visualização de dados por tipo e status

Design Responsivo: Layout que se adapta a diferentes telas

3. Gestão de Estado de UI
Gráficos: Controle de instâncias para evitar memory leaks

Tabelas Dinâmicas: Renderização eficiente de listas

Tema Escuro: Cores consistentes com a identidade visual

4. Fluxo de Dados

Firestore → Processamento → Renderização → Atualização UI
  ↓   ↓   ↓   ↓
 Recursos → Agregações → HTML/Charts → Cards/Tabelas

Características Técnicas:
✅ Componentização de templates HTML

✅ Separação entre dados e apresentação

✅ Tratamento de estados vazios

✅ Interface acessível (cores contrastantes)

✅ Performance (destruição de gráficos antigos)

Este código fornece uma base sólida para uma interface de administração moderna, responsiva e segura, com controle de acesso granular e visualizações de dados eficazes.

#login.html

1. Design e Layout
Layout Centrado: Interface focada no formulário de login

Tema Escuro Consistente: Mesma identidade visual da aplicação principal

Card Container: Agrupa todos os elementos do login de forma organizada

Responsivo: Adapta-se a diferentes tamanhos de tela

2. Elementos de Marca
Logo da Empresa: Identidade visual clara

Título Descritivo: "Acesso ao Sistema"

Nome da Empresa: "Indústrias Wayne" com destaque

3. Formulário de Login
Campos Obrigatórios: Email e senha com validação

Tipos Específicos: email para validação nativa, password para ocultação

Botão de Ação Primária: Destaque visual para a ação principal

Feedback Visual: Campos com styling do tema escuro

4. Tratamento de Erros
Container Dedicado: Área específica para mensagens de erro

Oculto Inicialmente: Só aparece quando necessário

Estilo de Alerta: Cores e ícones apropriados para erros

5. Acessibilidade
VLibras Integrado: Suporte a Libras brasileira

Labels Appropriados: Elementos label conectados aos inputs

Contraste Adequado: Cores com boa legibilidade

6. Performance e Segurança
Scripts Mínimos: Apenas o necessário para autenticação

Firebase Auth: Autenticação segura e gerenciada

Carregamento Ordenado: Dependências na ordem correta

Fluxo do Usuário:

Acessa Página → Visualiza Formulário → Preencre Credenciais →
Submete → (Sucesso) Redireciona para App → (Erro) Mostra Mensagem

Características de Segurança:
✅ Validação de campos no cliente

✅ Autenticação via Firebase Auth

✅ Mensagens de erro genéricas (não revelam detalhes de segurança)

✅ Campos password com type apropriado

✅ HTTPS implícito com CDNs

Esta página fornece uma experiência de login segura, acessível e consistente com a identidade visual da aplicação principal.

#index.html

1. Cabeçalho e Metadados
Configurações de Viewport: Design responsivo para mobile

Favicon: Identidade visual da empresa

Título da Página: "Indústrias Wayne - Painel de Gerenciamento"

2. Estrutura de Navegação
Branding: Logo e nome da empresa

Menu Principal: Dashboard e Recursos (com controle de acesso)

Área do Usuário: Email e botão de logout

3. Área de Conteúdo Dinâmico
Container Principal: Onde toda a aplicação é renderizada

Injeção Dinâmica: Conteúdo gerado via JavaScript

4. Componentes Modais
Modal de Recursos: Formulário reutilizável para criar/editar recursos

Tema Escuro: Consistente com o design da aplicação

Campos: Nome, Tipo, Status com validação

5. Acessibilidade
VLibras: Integração com tradução para Libras brasileira

Atributos ALT: Textos alternativos para imagens

6. Dependências e Scripts
Bootstrap: Styling e componentes UI

Chart.js: Visualizações de dados

Firebase: Backend como serviço (Auth, Firestore, Storage)

Scripts Customizados: Lógica da aplicação em módulos separados

Características de Design:
✅ Tema Escuro: Interface moderna e reduz fadiga visual

✅ Responsividade: Funciona em desktop, tablet e mobile

✅ Hierarquia Visual: Cores e espaçamentos consistentes

✅ Acessibilidade: Suporte a leitores de tela e Libras

✅ Performance: Carregamento otimizado de recursos

Esta estrutura fornece uma base sólida para uma aplicação web empresarial moderna, segura e acessível.