**ROADMAP**

**🔹 Roadmap para Extensão**

📌 **Fase 1 - Planejamento e Estruturação**

* Criar a estrutura do projeto
* Definir como será feita a identificação dos campos
* Escolher bibliotecas para geração de dados

📌 **Fase 2 - Captura e Classificação dos Campos**

* Criar o script no content.ts para detectar os formulários
* Implementar um sistema para reconhecer os tipos de dados

📌 **Fase 3 - UI e Funcionalidade de Preenchimento**

* Criar a interface para visualizar os campos encontrados
* Implementar a funcionalidade de preenchimento automático

📌 **Fase 4 - Salvamento e Recuperação de Dados**

* Salvar os dados do usuário no Chrome Storage ou Firebase
* Criar um botão para carregar os últimos dados salvos

📌 **Fase 5 - Monetização e Assinaturas**

* Implementar pagamentos via PayPal e Stripe
* Criar sistema de bloqueio de funcionalidades para não assinantes

📌 **Fase 6 - Testes e Publicação**

* Testar a extensão em diferentes sites
* Criar um plano de lançamento na Chrome Web Store

**PLANO DE IMPLEMENTAÇÃO**

**📌 1. Estruturação do Projeto e Planejamento**

🔹 **Definir funcionalidades principais** (feito!)  
🔹 **Organizar os arquivos** corretamente dentro da estrutura do projeto  
🔹 **Criar um roadmap** para seguir na implementação

**📌 2. Identificação e Extração de Campos do Formulário**

🔹 Criar um **script no content.ts** para varrer a página e identificar os campos (input, textarea, select, checkbox, etc.)  
🔹 Desenvolver uma **função para classificar cada campo**, reconhecendo se é nome, CPF, endereço, etc. (podemos usar heurísticas e regex para isso)  
🔹 Testar a captura e categorização dos campos

**📌 3. Criar a Interface para Gerenciar os Campos no Popup**

🔹 Criar a **tela de gerenciamento dos campos** encontrados na página  
🔹 Exibir os campos detectados e permitir que o usuário defina valores fixos ou use dados aleatórios  
🔹 Adicionar um botão para **preencher os campos automaticamente**

**📌 4. Gerar Dados Fictícios para os Campos**

🔹 Escolher e integrar **bibliotecas para geração de dados aleatórios**, como:

* [Faker.js](https://fakerjs.dev/) para nome, endereço, telefone, etc.
* [CPF-CNPJ Validator](https://www.npmjs.com/package/cpf-cnpj-validator) para CPF/CNPJ
* [Creditcard-generator](https://www.npmjs.com/package/creditcard-generator) para cartões de crédito
* Geradores de CEP e RG personalizados  
  🔹 Criar uma função que gere os dados certos para cada campo com base no seu tipo

**📌 5. Salvar e Recuperar as Informações do Usuário**

🔹 **Salvar as informações** preenchidas pelo usuário na extensão para uso posterior  
🔹 Usar o **Chrome Storage API** (para salvar localmente) ou Firebase Free (caso precise salvar na nuvem para múltiplos dispositivos)  
🔹 Permitir que o usuário escolha entre **usar os últimos dados salvos ou gerar novos valores**

**📌 6. Implementar a Funcionalidade de Preenchimento Automático**

🔹 Criar um botão **"Preencher Campos"** que recupera os dados salvos ou gera novos  
🔹 Fazer a injeção dos valores no formulário  
🔹 Testar em diferentes sites

**📌 7. Criar um Sistema de Assinatura e Monetização**

🔹 **Decidir o modelo de assinatura** (mensal, trimestral, anual) com teste grátis de 7 dias  
🔹 Criar um **sistema de pagamentos com PayPal e Stripe**:

* Configurar webhook do PayPal e Stripe para gerenciar pagamentos automaticamente
* Salvar status da assinatura do usuário no Firebase (caso use um banco)  
  🔹 Criar **bloqueios de funcionalidades para usuários não pagantes**

**📌 8. Ajustes Finais e Publicação**

🔹 Implementar **melhorias na UI/UX**  
🔹 Criar **configurações para o usuário** ajustar preferências  
🔹 Fazer **testes extensivos** para garantir funcionamento em diferentes sites  
🔹 Publicar na **Chrome Web Store** 🎉

**🔥 Sugestão para Gestão de Custos e Monetização**

✅ Use **Firebase Free** para armazenar informações de usuários até ter uma base sólida  
✅ Para pagamentos, comece apenas com **PayPal (sem precisar de CNPJ)** e depois adicione Stripe  
✅ Após validar a demanda, registre um **CNPJ MEI** e invista em **um plano Firebase pago** caso necessário

**Passos seguintes:**

Temos um desafio bem interessante aqui! O fluxo da funcionalidade pode ser dividido em três partes principais:

### 📌 ****1. Captura e Exibição dos Campos****

Opções para exibir os campos capturados:

* **Exibir na janela da extensão** (popup): Mais acessível, mas pode exigir um scroll longo se houver muitos campos.
* **Abrir uma nova aba** (chrome.tabs.create): Melhor para formulários longos, pois permite mais espaço.
* **Modal flutuante dentro da página** (injetado via content script): Mantém o usuário na página, mas pode ser intrusivo.

### 📌 ****2. Escolha entre Preenchimento Automático ou Manual****

* Adicionar um **switch toggle (checkbox estilizado)** para o usuário escolher entre preencher automaticamente ou manualmente.
* Se "Automático" for selecionado, os campos são preenchidos ao carregar a página.
* O usuário pode editar qualquer campo após o preenchimento automático.

### 📌 ****3. Salvamento dos Dados no Navegador****

Cada página/formulário deve ter uma **chave única baseada no domínio e na estrutura dos campos** para armazenar os valores:

* **Armazenamento no chrome.storage.local** → Dados persistem na extensão.
* **Formato do objeto salvo:**
* {
* "https://exemplo.com/formulario": {
* "fields": [
* {"id": "name", "value": "João Silva"},
* {"id": "email", "value": "joao@email.com"}
* ],
* "autoFill": true
* }
* }
* Assim, sempre que o usuário visitar o mesmo formulário, a extensão pode recuperar os dados.

### 🚀 ****Como implementar isso?****

#### **1️⃣ Capturar e exibir os campos**

No **content script**, capturamos os campos e enviamos para o popup da extensão:

const capturedFields = [...document.querySelectorAll("input, select, textarea")].map(field => ({

id: field.id || field.name || "",

placeholder: field.placeholder || "",

type: field.type || "text",

value: field.value || "",

category: "desconhecido" // Melhorar com uma lógica

}));

chrome.runtime.sendMessage({ action: "sendFields", fields: capturedFields });

#### **2️⃣ Exibir os campos no popup**

No **popup.html**, renderizamos os campos capturados e adicionamos um toggle para ativar/desativar o preenchimento automático.

<label>

<input type="checkbox" id="autoFillToggle">

Preenchimento Automático

</label>

<div id="fieldsContainer"></div>

<button id="saveFields">Salvar Dados</button>

E no **popup.js**, renderizamos os campos:

chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {

if (message.action === "sendFields") {

const container = document.getElementById("fieldsContainer");

container.innerHTML = ""; // Limpa antes de exibir

message.fields.forEach(field => {

const fieldDiv = document.createElement("div");

fieldDiv.innerHTML = `

<label>${field.placeholder || field.id}</label>

<input type="${field.type}" value="${field.value}" data-id="${field.id}">

`;

container.appendChild(fieldDiv);

});

document.getElementById("saveFields").addEventListener("click", () => {

const fieldsData = [...container.querySelectorAll("input")].map(input => ({

id: input.dataset.id,

value: input.value

}));

chrome.storage.local.set({ [window.location.hostname]: { fields: fieldsData, autoFill: document.getElementById("autoFillToggle").checked } });

});

}

});

#### **3️⃣ Preencher automaticamente**

Criamos um **content script** que verifica se há dados salvos para o domínio atual:

chrome.storage.local.get(window.location.hostname, (data) => {

if (data[window.location.hostname]?.autoFill) {

data[window.location.hostname].fields.forEach(field => {

let input = document.getElementById(field.id);

if (input) input.value = field.value;

});

}

});

### 🎯 ****Conclusão****

✅ **Exibe os campos no popup da extensão**  
✅ **Permite salvar e recuperar os valores**  
✅ **Oferece preenchimento automático opcional**  
✅ **Usuário pode editar os campos após preenchê-los**

Agora, só falta decidir como exibir os campos (popup, aba nova ou modal). Como prefere? 🔥