

Aplicativo de Lembrete para Beber Água - Arquivo Único

Este arquivo contém todo o código e instruções necessárias para criar e executar o aplicativo de lembrete para beber água. Você pode usar outra IA para criar os diretórios e executar o aplicativo com base nessas informações.

Estrutura de Diretórios

```
Manos/
├── src/
│   ├── hydration_reminder.py
│   ├── water_detector.py
│   └── requirements.txt
├── README.md
└── mac_adaptation_instructions.md
```

Comandos para Configuração

```
# Criar diretórios
mkdir -p ~/Desktop/Manos/src

# Instalar dependências
cd ~/Desktop/Manos
pip3 install -r src/requirements.txt

# Executar o aplicativo
python3 src/hydration_reminder.py
```

Conteúdo dos Arquivos

src/hydration_reminder.py

```
"""
Aplicativo de Lembrete para Beber Água
Protótipo Cross-Platform com Instruções para Adaptação no Mac

Este aplicativo lembra o usuário de beber água periodicamente e utiliza
```

a câmera para detectar quando o usuário está bebendo água (copo ou garrafa).
"""

```
import os
import time
import threading
import tkinter as tk
from tkinter import messagebox, simpledialog
import cv2
import numpy as np
from datetime import datetime, timedelta
```

Importar o detector de água

```
from water_detector import WaterContainerDetector
```

```
class HydrationReminder:
```

```
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Lembrete para Beber Água")
        self.root.geometry("400x300")
        self.root.resizable(False, False)
```

Configurações padrão

```
self.interval_minutes = 30
self.is_active = False
self.next_reminder = None
self.reminder_thread = None
self.camera_thread = None
self.stop_camera = False
self.water_detected = False
```

Inicializar o detector de água

```
self.water_detector = WaterContainerDetector()
```

```
self.setup_ui()
```

```
def setup_ui(self):
```

Título

```
    title_label = tk.Label(self.root, text="Lembrete para Beber Água", font=("Arial",
16, "bold"))
    title_label.pack(pady=20)
```

Configuração do intervalo

```
    interval_frame = tk.Frame(self.root)
    interval_frame.pack(pady=10)
```

```
    interval_label = tk.Label(interval_frame, text="Intervalo (minutos):")
    interval_label.pack(side=tk.LEFT, padx=5)
```

```
    self.interval_var = tk.StringVar(value=str(self.interval_minutes))
    interval_entry = tk.Entry(interval_frame, textvariable=self.interval_var, width=5)
    interval_entry.pack(side=tk.LEFT, padx=5)
```

```

# Botões
button_frame = tk.Frame(self.root)
button_frame.pack(pady=20)

self.start_button = tk.Button(button_frame, text="Iniciar",
command=self.start_reminder, width=10)
self.start_button.pack(side=tk.LEFT, padx=10)

self.stop_button = tk.Button(button_frame, text="Parar",
command=self.stop_reminder, width=10, state=tk.DISABLED)
self.stop_button.pack(side=tk.LEFT, padx=10)

# Botão de teste da câmera
self.test_camera_button = tk.Button(button_frame, text="Testar Câmera",
command=self.test_camera, width=12)
self.test_camera_button.pack(side=tk.LEFT, padx=10)

# Status
self.status_frame = tk.Frame(self.root)
self.status_frame.pack(pady=10, fill=tk.X, padx=20)

self.status_label = tk.Label(self.status_frame, text="Status: Inativo",
font=("Arial", 10))
self.status_label.pack(side=tk.LEFT)

self.next_reminder_label = tk.Label(self.status_frame, text="", font=("Arial", 10))
self.next_reminder_label.pack(side=tk.RIGHT)

# Informações
info_text = "Este aplicativo irá lembrá-lo de beber água periodicamente.\n"
info_text +=
"Quando for notificado, beba água e mostre o copo/garrafa para a câmera."
info_label = tk.Label(self.root, text=info_text, justify=tk.LEFT, wraplength=380)
info_label.pack(pady=20)

def start_reminder(self):
    try:
        self.interval_minutes = int(self.interval_var.get())
        if self.interval_minutes <= 0:
            messagebox.showerror("Erro", "O intervalo deve ser maior que zero.")
            return
    except ValueError:
        messagebox.showerror("Erro", "Por favor, insira um número válido para o
intervalo.")
        return

self.is_active = True
self.start_button.config(state=tk.DISABLED)
self.stop_button.config(state=tk.NORMAL)
self.test_camera_button.config(state=tk.DISABLED)
self.status_label.config(text="Status: Ativo")

```

```

        self.next_reminder = datetime.now() +
timedelta(minutes=self.interval_minutes)
        self.update_next_reminder_label()

# Iniciar thread de lembrete
if self.reminder_thread is None or not self.reminder_thread.is_alive():
    self.reminder_thread = threading.Thread(target=self.reminder_loop)
    self.reminder_thread.daemon = True
    self.reminder_thread.start()

def stop_reminder(self):
    self.is_active = False
    self.start_button.config(state=tk.NORMAL)
    self.stop_button.config(state=tk.DISABLED)
    self.test_camera_button.config(state=tk.NORMAL)
    self.status_label.config(text="Status: Inativo")
    self.next_reminder_label.config(text="")

def update_next_reminder_label(self):
    if self.next_reminder and self.is_active:
        time_str = self.next_reminder.strftime("%H:%M:%S")
        self.next_reminder_label.config(text=f"Próximo: {time_str}")
        self.root.after(1000, self.update_next_reminder_label)

def reminder_loop(self):
    while self.is_active:
        current_time = datetime.now()
        if current_time >= self.next_reminder:
            self.show_reminder()
            # Próximo lembrete será definido após a detecção de água
            time.sleep(1)
        else:
            time.sleep(1)

def show_reminder(self):
    # Criar uma nova janela de lembrete que fica por cima de tudo
    self.reminder_window = tk.Toplevel(self.root)
    self.reminder_window.title("Hora de Beber Água!")
    self.reminder_window.attributes('-topmost', True) # Mantém a janela no topo

    # Em sistemas Mac, seria usado PyObjC para criar uma janela de bloqueio total

    # Configurar a janela para ocupar toda a tela
    screen_width = self.reminder_window.winfo_screenwidth()
    screen_height = self.reminder_window.winfo_screenheight()
    self.reminder_window.geometry(f"{screen_width}x{screen_height}+0+0")

    # Adicionar conteúdo à janela
    frame = tk.Frame(self.reminder_window, bg="#e6f7ff", padx=50, pady=50)
    frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

```

```

message_label = tk.Label(
    frame,
    text="Hora de beber água!",
    font=("Arial", 24, "bold"),
    bg="#e6f7ff"
)
message_label.pack(pady=20)

instruction_label = tk.Label(
    frame,
    text="Beba água e mostre o copo/garrafa para a câmera para continuar.",
    font=("Arial", 14),
    wraplength=400,
    bg="#e6f7ff"
)
instruction_label.pack(pady=20)

# Status da detecção
self.detection_status = tk.Label(
    frame,
    text="Aguardando...",
    font=("Arial", 12),
    bg="#e6f7ff"
)
self.detection_status.pack(pady=10)

# Iniciar detecção de câmera
self.water_detected = False
self.stop_camera = False

# Iniciar thread da câmera
self.camera_thread = threading.Thread(target=self.detect_water_container)
self.camera_thread.daemon = True
self.camera_thread.start()

# Botão para fechar manualmente (para testes)
skip_button = tk.Button(
    frame,
    text="Pular (apenas para teste)",
    command=self.manual_close_reminder
)
skip_button.pack(pady=20)

def manual_close_reminder(self):
    self.stop_camera = True
    if self.camera_thread:
        self.camera_thread.join(timeout=1)
    self.close_reminder()

def close_reminder(self):
    if hasattr(self, 'reminder_window') and self.reminder_window:
        self.reminder_window.destroy()

```

```

    # Definir próximo lembrete
    self.next_reminder = datetime.now() +
timedelta(minutes=self.interval_minutes)

    def detect_water_container(self):
        """
        Função para detectar copos ou garrafas de água usando o módulo
        WaterContainerDetector.
        """
        try:
            # Inicializar a câmera
            cap = cv2.VideoCapture(0)
            if not cap.isOpened():
                print("Erro ao abrir a câmera")
                self.update_detection_status("Erro ao acessar a câmera")
                return

            self.update_detection_status("Câmera ativada. Procurando copo/garrafa...")

            detection_count = 0 # Contador para confirmar detecções consecutivas

            while not self.stop_camera:
                ret, frame = cap.read()
                if not ret:
                    break

                # Usar o detector de água
                detected, processed_frame =
self.water_detector.detect_water_container(frame)

                # Mostrar imagem para debug
                cv2.imshow('Detecção de Água', processed_frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break

                # Atualizar status
                if detected:
                    detection_count += 1
                    self.update_detection_status(f"Possível copo/garrafa detectado!
({detection_count}/3)")

                    # Confirmar detecção após 3 frames consecutivos
                    if detection_count >= 3:
                        self.update_detection_status("Copo/garrafa confirmado!
Liberando...")
                        self.water_detected = True
                        self.root.after(2000, self.close_reminder) # Fechar após 2 segundos
                        self.stop_camera = True
                        break
                else:
                    detection_count = 0

```

```

        self.update_detection_status("Procurando copo/garrafa...")

        time.sleep(0.1)

        # Liberar recursos
        cap.release()
        cv2.destroyAllWindows()

    except Exception as e:
        print(f"Erro na detecção: {e}")
        self.update_detection_status(f"Erro: {str(e)}")

    def update_detection_status(self, text):
        """Atualiza o texto de status da detecção na interface."""
        if hasattr(self, 'detection_status') and self.detection_status:
            self.detection_status.config(text=text)

    def test_camera(self):
        """Função para testar a detecção de câmera."""
        test_window = tk.Toplevel(self.root)
        test_window.title("Teste de Câmera")
        test_window.geometry("500x400")

        info_label = tk.Label(
            test_window,
            text="Teste a detecção de copos/garrafas.
\nMostre um copo ou garrafa para a câmera.",
            font=("Arial", 12),
            wraplength=450
        )
        info_label.pack(pady=20)

        status_label = tk.Label(
            test_window,
            text="Iniciando câmera...",
            font=("Arial", 10)
        )
        status_label.pack(pady=10)

        close_button = tk.Button(
            test_window,
            text="Fechar Teste",
            command=lambda: self.stop_test_camera(test_window)
        )
        close_button.pack(pady=10)

        # Iniciar thread de teste
        self.stop_camera = False
        self.test_thread = threading.Thread(
            target=self.run_camera_test,
            args=(status_label,)
        )

```

```

self.test_thread.daemon = True
self.test_thread.start()

def run_camera_test(self, status_label):
    """Executa o teste de câmera."""
    try:
        # Inicializar a câmera
        cap = cv2.VideoCapture(0)
        if not cap.isOpened():
            status_label.config(text="Erro ao abrir a câmera")
            return

        status_label.config(text="Câmera ativada. Procurando copo/garrafa...")

        while not self.stop_camera:
            ret, frame = cap.read()
            if not ret:
                break

            # Usar o detector de água
            detected, processed_frame =
self.water_detector.detect_water_container(frame)

            # Mostrar imagem
            cv2.imshow('Teste de Detecção', processed_frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

            # Atualizar status
            if detected:
                status_label.config(text="Copo/garrafa detectado!")
            else:
                status_label.config(text="Procurando copo/garrafa...")

            time.sleep(0.1)

            # Liberar recursos
            cap.release()
            cv2.destroyAllWindows()

        except Exception as e:
            print(f"Erro no teste de câmera: {e}")
            status_label.config(text=f"Erro: {str(e)}")

def stop_test_camera(self, test_window):
    """Para o teste de câmera e fecha a janela."""
    self.stop_camera = True
    if hasattr(self, 'test_thread') and self.test_thread.is_alive():
        self.test_thread.join(timeout=1)
    cv2.destroyAllWindows()
    test_window.destroy()

```



```

def run(self):
    self.root.mainloop()

if __name__ == "__main__":
    app = HydrationReminder()
    app.run()

```

src/water_detector.py

```

"""
Módulo de detecção de copos e garrafas de água usando OpenCV.
Este módulo fornece funcionalidades para detectar copos e garrafas de água usando a
câmera.
"""

import cv2
import numpy as np
import time
import os

class WaterContainerDetector:
    def __init__(self):
        # Diretório para armazenar o modelo YOLO (se disponível)
        self.model_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)),
'models')
        os.makedirs(self.model_dir, exist_ok=True)

        # Flag para indicar se estamos usando detecção simples ou avançada
        self.use_advanced_detection = False

        # Tentar carregar modelo YOLO se disponível
        self.net = None
        self.try_load_yolo_model()

    def try_load_yolo_model(self):
        """Tenta carregar o modelo YOLO para detecção avançada de objetos."""
        try:
            # Verificar se os arquivos do modelo existem
            config_path = os.path.join(self.model_dir, 'yolov3.cfg')
            weights_path = os.path.join(self.model_dir, 'yolov3.weights')
            classes_path = os.path.join(self.model_dir, 'coco.names')

            if os.path.exists(config_path) and os.path.exists(weights_path) and
os.path.exists(classes_path):
                # Carregar modelo YOLO
                self.net = cv2.dnn.readNetFromDarknet(config_path, weights_path)

                # Carregar classes
                with open(classes_path, 'r') as f:
                    self.classes = [line.strip() for line in f.readlines()]

```

```

# Configurar backend
self.net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

self.use_advanced_detection = True
print("Modelo YOLO carregado com sucesso!")
else:
    print("Arquivos do modelo YOLO não encontrados. Usando detecção
simples.")
except Exception as e:
    print(f"Erro ao carregar modelo YOLO: {e}")
    print("Usando detecção simples baseada em cor e forma.")

def detect_water_container(self, frame):
    """
    Detecta copos ou garrafas de água no frame da câmera.

    Args:
        frame: Frame da câmera (imagem OpenCV)

    Returns:
        bool: True se um copo ou garrafa for detectado, False caso contrário
        frame: Frame com as detecções marcadas
    """
    if self.use_advanced_detection and self.net is not None:
        return self.detect_with_yolo(frame)
    else:
        return self.detect_with_color_and_shape(frame)

def detect_with_yolo(self, frame):
    """
    Detecta copos ou garrafas usando o modelo YOLO.

    Args:
        frame: Frame da câmera

    Returns:
        bool: True se um copo ou garrafa for detectado, False caso contrário
        frame: Frame com as detecções marcadas
    """
    height, width, _ = frame.shape

    # Criar blob a partir da imagem
    blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True,
crop=False)
    self.net.setInput(blob)

    # Obter camadas de saída
    layer_names = self.net.getLayerNames()
    output_layers = [layer_names[i - 1] for i in
self.net.getUnconnectedOutLayers()]

```

```

# Executar detecção
outputs = self.net.forward(output_layers)

# Inicializar listas
class_ids = []
confidences = []
boxes = []

# Para cada detecção
for output in outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]

        # Filtrar detecções relevantes (copo, garrafa, etc.)
        relevant_classes = ['cup', 'bottle', 'wine glass', 'glass']
        if confidence > 0.5 and self.classes[class_id] in relevant_classes:
            # Coordenadas do objeto
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Coordenadas do retângulo
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

# Aplicar non-maximum suppression
indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

# Desenhar caixas delimitadoras
detected = False
for i in range(len(boxes)):
    if i in indexes:
        detected = True
        x, y, w, h = boxes[i]
        label = f"{self.classes[class_ids[i]]}: {confidences[i]:.2f}"
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
255, 0), 2)

return detected, frame

def detect_with_color_and_shape(self, frame):
    """
    Detecta copos ou garrafas usando técnicas simples de cor e forma.

```

Esta é uma implementação simplificada para o protótipo.

Args:

frame: Frame da câmera

Returns:

bool: True se um copo ou garrafa for detectado, False caso contrário

frame: Frame com as detecções marcadas

"""

Converter para HSV para facilitar a detecção de cor

hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

Definir faixas de cor para detecção

Azul claro/transparente para água

lower_blue = np.array([90, 50, 50])

upper_blue = np.array([130, 255, 255])

Transparente/claro para copos de vidro

lower_clear = np.array([0, 0, 150])

upper_clear = np.array([180, 30, 255])

Criar máscaras

mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)

mask_clear = cv2.inRange(hsv, lower_clear, upper_clear)

Combinar máscaras

mask = cv2.bitwise_or(mask_blue, mask_clear)

Aplicar operações morfológicas para remover ruído

kernel = np.ones((5, 5), np.uint8)

mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

Encontrar contornos

contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

Verificar se há contornos grandes o suficiente (possível copo/garrafa)

detected = **False**

for contour **in** contours:

area = cv2.contourArea(contour)

if area > 3000: *# Ajustar este valor conforme necessário*

Calcular proporção altura/largura para identificar formas de copo/garrafa

x, y, w, h = cv2.boundingRect(contour)

aspect_ratio = **float**(h) / w

Copos e garrafas geralmente têm proporção altura/largura > 1

if aspect_ratio > 1.2:

Possível detecção de copo/garrafa

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.putText(frame, "**Copo/Garrafa**", (x, y - 10),

cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

```

        detected = True

        # Adicionar informações de debug na imagem
        cv2.putText(frame, "Detecção Simples (Cor e Forma)", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        status = "Detectado" if detected else "Não Detectado"
        cv2.putText(frame, f'Status: {status}', (10, 60), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 0, 255), 2)

        return detected, frame

def test_camera_detection():
    """Função para testar a detecção de câmera."""
    detector = WaterContainerDetector()
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("Erro ao abrir a câmera")
        return

    print("Pressione 'q' para sair")

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Detectar copo/garrafa
        detected, processed_frame = detector.detect_water_container(frame)

        # Mostrar resultado
        cv2.imshow('Detecção de Copo/Garrafa', processed_frame)

        # Sair com a tecla 'q'
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

        time.sleep(0.03) # Limitar a ~30 FPS

    # Liberar recursos
    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    test_camera_detection()

```

src/requirements.txt

```

opencv-python>=4.5.0
numpy>=1.20.0

```

```
pillow>=8.0.0
tkinter
# Para adaptação no Mac
# pyobjc>=7.3
# rumps>=0.3.0
```

README.md

Lembrete para Beber Água - README

Este é um aplicativo que lembra você de beber água periodicamente e utiliza a câmera para detectar quando você está bebendo água (copo ou garrafa).

Funcionalidades

- Lembretes periódicos para beber água
- Bloqueio de tela/overlay quando é hora de beber água
- Detecção de copos e garrafas de água usando a câmera
- Desbloqueio automático quando detecta que você bebeu água
- Interface gráfica simples para configuração

Requisitos

- Python 3.7 ou superior
- OpenCV
- NumPy
- Tkinter
- Acesso à câmera

Instalação

1. Clone ou baixe este repositório
2. Instale as dependências:

```
""
pip install -r src/requirements.txt
""
```

Uso

1. Execute o aplicativo:

```
""
python src/hydration_reminder.py
""
```

2. Configure o intervalo de tempo entre os lembretes (em minutos)
3. Clique em "Iniciar" para começar os lembretes
4. Quando aparecer o lembrete, beba água e mostre o copo/garrafa para a câmera
5. O aplicativo detectará automaticamente e liberará a tela

Adaptação para macOS

Para instruções detalhadas sobre como adaptar este aplicativo para macOS, incluindo bloqueio de tela completo e integração com a barra de menu, consulte o arquivo ``mac_adaptation_instructions.md``.

Estrutura do Projeto

- ``src/hydration_reminder.py``: Aplicativo principal
- ``src/water_detector.py``: Módulo de detecção de copos/garrafas
- ``src/requirements.txt``: Dependências do projeto
- ``mac_adaptation_instructions.md``: Instruções para adaptação no macOS

Limitações do Protótipo

- A detecção de copos/garrafas é baseada em técnicas simples de cor e forma
- Para uma detecção mais precisa, seria necessário um modelo de aprendizado de máquina treinado

Próximos Passos

- Implementar um modelo de detecção mais avançado (YOLO, SSD)
- Adicionar estatísticas de consumo de água
- Personalização de sons e mensagens

mac_adaptation_instructions.md

Instruções para Adaptação no macOS

Este documento fornece instruções detalhadas para adaptar o protótipo do aplicativo de lembrete para beber água para o macOS, incluindo as modificações necessárias para implementar o bloqueio de tela completo e outras funcionalidades específicas do Mac.

Requisitos do Sistema

- macOS 10.14 (Mojave) ou superior
- Python 3.7 ou superior
- Permissões de acesso à câmera

Instalação de Dependências

1. Abra o Terminal e navegue até o diretório do projeto:

```
```bash
cd /caminho/para/hydration_reminder
```
```

2. Instale as dependências específicas para Mac:

```
```bash
pip3 install -r src/requirements.txt
pip3 install pyobjc rumps
```
```

Modificações Necessárias para o macOS

1. Bloqueio de Tela Completo

Para implementar o bloqueio de tela completo no macOS, é necessário modificar o código para usar o PyObjC. Crie um novo arquivo chamado `mac_screen_blocker.py` com o seguinte conteúdo:

```
```python
"""
Módulo para bloqueio de tela no macOS usando PyObjC.
"""
import Cocoa
import objc
from Foundation import NSObject, NSTimer
import time

class ScreenBlocker(NSObject):
 def init(self):
 self = objc.super(ScreenBlocker, self).init()
 if self is None:
 return None

 self.app = Cocoa.NSApplication.sharedApplication()
 self.windows = []
 return self

 def block_screen(self, message="Hora de beber água!", callback=None):
 """
 Bloqueia a tela com uma janela em cada monitor.

 Args:
 message: Mensagem a ser exibida
 callback: Função a ser chamada quando o bloqueio for removido
 """
 # Obter todos os monitores
 screens = Cocoa.NSScreen.screens()

 # Criar uma janela para cada monitor
 for screen in screens:
 # Criar janela
 window = Cocoa.NSWindow.alloc()
 rect = screen.frame()
 window initWithContentRect_styleMask_backing_defer_(
 rect,
 Cocoa.NSWindowStyleMaskBorderless,
 Cocoa.NSBackingStoreBuffered,
 False
)

```



```
Configurar janela

window.setBackgroundColor_(Cocoa.NSColor.colorWithCalibratedRed_green_blue_alpha_(
 0.9, 0.95, 1.0, 0.9)) # Azul claro semi-transparente
window.setLevel_(Cocoa.NSScreenSaverWindowLevel) # Nível acima de
todas as janelas
window.setOpaque_(False)
window.setHasShadow_(False)

Adicionar conteúdo
content_view = window.contentView()

Adicionar mensagem
text_field = Cocoa.NSTextField.alloc().initWithFrame_(
 Cocoa.NSMakeRect(0, rect.size.height / 2, rect.size.width, 50)
)
text_field.setStringValue_(message)
text_field.setAlignment_(Cocoa.NSTextAlignmentCenter)
text_field.setFont_(Cocoa.NSFont.boldSystemFontOfSize_(36))
text_field.setTextColor_(Cocoa.NSColor.blackColor())
text_field.setBezeled_(False)
text_field.setDrawsBackground_(False)
text_field.setEditable_(False)
text_field.setSelectable_(False)

content_view.addSubview_(text_field)

Adicionar instrução
instruction = Cocoa.NSTextField.alloc().initWithFrame_(
 Cocoa.NSMakeRect(0, rect.size.height / 2 - 50, rect.size.width, 30)
)
instruction.setStringValue_("Beba água e mostre o copo/garrafa para a
câmera para continuar.")
instruction.setAlignment_(Cocoa.NSTextAlignmentCenter)
instruction.setFont_(Cocoa.NSFont.systemFontOfSize_(18))
instruction.setTextColor_(Cocoa.NSColor.blackColor())
instruction.setBezeled_(False)
instruction.setDrawsBackground_(False)
instruction.setEditable_(False)
instruction.setSelectable_(False)

content_view.addSubview_(instruction)

Mostrar janela
window.makeKeyAndOrderFront_(None)

Armazenar referência à janela
self.windows.append(window)

Iniciar loop de eventos
self.app.run()
```

```

Armazenar callback
self.callback = callback

def unblock_screen(self):
 """Remove o bloqueio de tela."""
 # Fechar todas as janelas
 for window in self.windows:
 window.close()

 self.windows = []

 # Parar loop de eventos
 self.app.stop_(None)

 # Chamar callback se existir
 if hasattr(self, 'callback') and self.callback:
 self.callback()

Exemplo de uso
if __name__ == "__main__":
 blocker = ScreenBlocker.alloc().init()

 def on_unblock():
 print("Tela desbloqueada!")

 # Bloquear tela por 5 segundos
 import threading

 def unblock_after_delay():
 time.sleep(5)
 blocker.unblock_screen()

 threading.Thread(target=unblock_after_delay).start()
 blocker.block_screen(callback=on_unblock)

```

## 2. Integração com Menu de Status do macOS

Para adicionar um ícone na barra de menu do macOS, crie um arquivo chamado `mac_status_menu.py` :

```

"""
Módulo para criar um ícone na barra de menu do macOS usando rumps.
"""

import rumps
import threading
import time
from datetime import datetime, timedelta

class HydrationStatusBarApp(rumps.App):

```

```

def __init__(self, hydration_app):
 super(HydrationStatusBarApp, self).__init__(" ", quit_button=None)

 self.hydration_app = hydration_app
 self.timer = None
 self.next_reminder = None

 # Configurar menu
 self.menu = [
 rumps.MenuItem("Status: Inativo"),
 None, # Separador
 rumps.MenuItem("Iniciar", callback=self.start),
 rumps.MenuItem("Parar", callback=self.stop),
 rumps.MenuItem("Configurações", callback=self.show_settings),
 None, # Separador
 rumps.MenuItem("Sair", callback=self.quit_app)
]

 # Desativar botão "Parar" inicialmente
 self.menu["Parar"].set_callback(None)

def start(self, _):
 """Inicia o lembrete de hidratação."""
 # Chamar método de início do aplicativo principal
 self.hydration_app.start_reminder()

 # Atualizar menu
 self.menu["Status: Inativo"].title = "Status: Ativo"
 self.menu["Iniciar"].set_callback(None)
 self.menu["Parar"].set_callback(self.stop)

 # Iniciar timer para atualizar o título
 self.update_title()

def stop(self, _):
 """Para o lembrete de hidratação."""
 # Chamar método de parada do aplicativo principal
 self.hydration_app.stop_reminder()

 # Atualizar menu
 self.menu["Status: Ativo"].title = "Status: Inativo"
 self.menu["Iniciar"].set_callback(self.start)
 self.menu["Parar"].set_callback(None)

 # Parar timer
 self.title = " "

def show_settings(self, _):
 """Mostra a janela de configurações."""
 # Mostrar a janela principal do aplicativo
 self.hydration_app.root.deiconify()

```

```

def quit_app(self, _):
 """Sai do aplicativo."""
 rumps.quit_application()

def update_title(self):
 """Atualiza o título com o tempo restante."""
 if self.hydration_app.is_active and self.hydration_app.next_reminder:
 # Calcular tempo restante
 now = datetime.now()
 remaining = self.hydration_app.next_reminder - now

 if remaining.total_seconds() > 0:
 # Formatar tempo restante
 minutes, seconds = divmod(int(remaining.total_seconds()), 60)
 self.title = f" {minutes:02d}:{seconds:02d}"
 else:
 self.title = " Agora!"

 # Agendar próxima atualização
 threading.Timer(1.0, self.update_title).start()
 else:
 self.title = " "

Exemplo de uso
if __name__ == "__main__":
 # Criar um objeto fictício para teste
 class DummyApp:
 def __init__(self):
 self.is_active = False
 self.next_reminder = None
 self.root = None

 def start_reminder(self):
 self.is_active = True
 self.next_reminder = datetime.now() + timedelta(minutes=1)

 def stop_reminder(self):
 self.is_active = False
 self.next_reminder = None

 dummy_app = DummyApp()
 app = HydrationStatusBarApp(dummy_app)
 app.run()

```

### 3. Modificações no Arquivo Principal

Modifique o arquivo `hydration_reminder.py` para integrar as funcionalidades específicas do Mac:

```

No início do arquivo, adicione:
import platform

Verificar se está rodando no macOS
is_mac = platform.system() == 'Darwin'

Importar módulos específicos do Mac se necessário
if is_mac:
 try:
 from mac_screen_blocker import ScreenBlocker
 from mac_status_menu import HydrationStatusBarApp
 has_mac_modules = True
 except ImportError:
 has_mac_modules = False
else:
 has_mac_modules = False

```

E na classe `HydrationReminder`, modifique o método `show_reminder`:

```

def show_reminder(self):
 if is_mac and has_mac_modules:
 # Usar bloqueio de tela nativo do Mac
 self.screen_blocker = ScreenBlocker.alloc().init()

 # Iniciar detecção de câmera
 self.water_detected = False
 self.stop_camera = False

 # Iniciar thread da câmera
 self.camera_thread = threading.Thread(target=self.detect_water_container)
 self.camera_thread.daemon = True
 self.camera_thread.start()

 # Bloquear tela
 self.screen_blocker.block_screen(
 message="Hora de beber água!",
 callback=lambda: self.root.after(0, self.close_reminder)
)
 else:
 # Usar implementação cross-platform (código existente)
 # ...

```

E no método `run`, adicione:

```

def run(self):
 if is_mac and has_mac_modules:
 # Criar aplicativo de barra de status
 self.status_app = HydrationStatusBarApp(self)

```

```
Iniciar aplicativo de barra de status em uma thread separada
threading.Thread(target=self.status_app.run, daemon=True).start()
```

```
Iniciar interface principal
self.root.mainloop()
```

## Criando um Aplicativo Distribuível para macOS

Para criar um aplicativo distribuível para macOS, você pode usar o `py2app`. Siga estas etapas:

1. Instale o `py2app`: `bash pip3 install py2app`
2. Crie um arquivo `setup.py` na raiz do projeto: `python from setuptools import setup`

```
APP = ['src/hydration_reminder.py'] DATA_FILES = [] OPTIONS = { 'argv_emulation': True,
'packages': ['cv2', 'numpy', 'tkinter'], 'includes': ['water_detector', 'mac_screen_blocker',
'mac_status_menu'], 'plist': { 'CFBundleName': 'Lembrete para Beber Água',
'CFBundleDisplayName': 'Lembrete para Beber Água', 'CFBundleIdentifier':
'com.example.hydrationreminder', 'CFBundleVersion': '1.0.0',
'CFBundleShortVersionString': '1.0.0', 'NSCameraUsageDescription': 'Este aplicativo usa
a câmera para detectar quando você está bebendo água.', 'LSUIElement': True, # Permite
que o aplicativo seja executado sem aparecer no Dock }, 'iconfile': 'icon.icns', # Substitua
por um arquivo de ícone real }
```

```
setup(app=APP, data_files=DATA_FILES, options={'py2app': OPTIONS},
setup_requires=['py2app'],)
```

1. Crie o aplicativo: `bash python3 setup.py py2app`
2. O aplicativo será criado na pasta `dist`.

## Permissões de Câmera no macOS

No macOS, o aplicativo precisará de permissão para acessar a câmera. Quando o aplicativo tentar acessar a câmera pela primeira vez, o sistema solicitará permissão ao usuário. Para garantir uma experiência suave:

1. Adicione uma descrição clara no `Info.plist` (já incluído no `setup.py` acima).
2. Informe o usuário que ele precisará conceder permissão de câmera quando solicitado.

# Execução Automática na Inicialização

Para configurar o aplicativo para iniciar automaticamente quando o usuário fizer login:

1. Vá para Preferências do Sistema > Usuários e Grupos > Itens de Login
2. Clique no botão "+" e adicione o aplicativo da pasta `dist`

## Solução de Problemas

### Problema: Aplicativo não consegue acessar a câmera

- Verifique as permissões de câmera em Preferências do Sistema > Segurança e Privacidade > Câmera
- Certifique-se de que o aplicativo está na lista e tem permissão concedida

### Problema: Bloqueio de tela não funciona corretamente

- Verifique se o PyObjC está instalado corretamente: `pip3 show pyobjc`
- Tente reinstalar: `pip3 install --upgrade pyobjc`

### Problema: Ícone da barra de menu não aparece

- Verifique se o rumps está instalado corretamente: `pip3 show rumps`
- Tente reinstalar: `pip3 install --upgrade rumps`

## Recursos Adicionais

- [Documentação do PyObjC](#)
- [Documentação do rumps](#)
- [Guia de empacotamento de aplicativos Python para macOS](#) ``