

# TESTmess - Specifiche Tecniche Progetto

SPECIFICHE TECNICHE DEL PROGETTO - TESTmess v2.2.6

DOCUMENTO DI REQUISITI E IMPLEMENTAZIONE

△ ISTRUZIONI CRITICHE PER L'AI

□ STATO IMPLEMENTAZIONE

Funzionalità Completate

Ultima Modifica

□ CONTESTO DEL PROGETTO

Obiettivo Generale

Architettura

Flusso Dati

□ FUNZIONALITÀ DA IMPLEMENTARE

FUNZIONE 1: MODIFICA EVENTI GOOGLE CALENDAR

Descrizione

Requisiti Tecnici

Criteri di Completamento

FUNZIONE 2: CREAZIONE BOZZE EMAIL AUTOMATICHE

Descrizione

Requisiti Tecnici

Criteri di Completamento

FUNZIONE 3: GESTIONE ALLEGATI EMAIL

Descrizione

Requisiti Tecnici

Criteri di Completamento

FUNZIONE 4: SEZIONE ALLEGATI CON TEMPLATE DEFAULT

Descrizione

Requisiti Tecnici

Criteri di Completamento

□ TESTING COMPLETO

Test Funzione 1: Google Calendar

Test Funzione 2: Gmail Bozze

Test Funzione 3: Allegati

Test Funzione 4: Template con Allegati

□ DEPLOYMENT CHECKLIST

Pre-Deployment

Cloudflare Setup

Deployment

Post-Deployment

Zapier Configuration

□ STRUTTURA FINALE PROGETTO

- ENVIRONMENT VARIABLES
  - Local Development (.dev.vars)
  - Production (Cloudflare Secrets)
- COME USARE QUESTO DOCUMENTO
  - Per una nuova sessione AI:
  - Per l'utente:
  - Per aggiornamenti:
- ☒ TROUBLESHOOTING
  - Errore: "Invalid credentials" Google API
  - Errore: "File too large" (> 10MB)
  - Errore: "Template not found"
  - Webhook Zapier non funziona
- METRICHE E MONITORING
  - Logging
  - Statistiche da Tracciare
- CHECKLIST FINALE
  - Funzionalità
  - UI/UX
  - Sicurezza
  - Performance
  - Documentazione
  - Testing
  - Deployment
- PROSSIMI PASSI SUGGERITI

## SPECIFICHE TECNICHE DEL PROGETTO - TESTmess v2.2.6

### DOCUMENTO DI REQUISITI E IMPLEMENTAZIONE

**VERSIONE:** 2.2.6

**DATA:** 2026-01-06

**PROGETTO:** webapp (TESTmess)

**PERCORSO:** /home/user/webapp/

---

### ⚠ ISTRUZIONI CRITICHE PER L'AI

**IMPORTANTE: LEGGI ATTENTAMENTE PRIMA DI PROCEDERE**

**1. QUESTO È UN DOCUMENTO DI STATO DEL PROGETTO**

- Contiene le specifiche complete del sistema esistente
- Contiene le nuove funzionalità da implementare

- NON ricreare il progetto da zero
  - Lavora sui file esistenti in /home/user/webapp/
2. **WORKFLOW OBBLIGATORIO**
- Leggi TUTTO questo documento prima di iniziare
  - Lavora su UNA funzionalità alla volta
  - Completa e testa ogni funzione prima di passare alla successiva
  - Aggiorna questo file dopo ogni implementazione completata
3. **GESTIONE DELLA MEMORIA**
- Questo file è la “memoria persistente” del progetto
  - Quando l’utente ricarica la cartella, leggi questo file per capire lo stato
  - Aggiorna la sezione “STATO IMPLEMENTAZIONE” dopo ogni modifica
  - Mantieni traccia di cosa è fatto e cosa manca
- 

## ☐ STATO IMPLEMENTAZIONE

### Funzionalità Completate

- ☐ Base progetto Hono + Cloudflare Pages
- ☐ Funzione 1: Modifica eventi Google Calendar
- ☐ Funzione 2: Creazione bozze email automatiche
- ☐ Funzione 3: Gestione allegati email
- ☐ Funzione 4: Sezione allegati con template default

### Ultima Modifica

- **Data:** [DA AGGIORNARE]
  - **Funzionalità:** [DA AGGIORNARE]
  - **Note:** [DA AGGIORNARE]
- 

## ☐ CONTESTO DEL PROGETTO

### Obiettivo Generale

Creare un’applicazione web su Cloudflare Pages che automatizzi la gestione di: - Eventi su Google Calendar - Bozze email su Gmail - Allegati email con template predefiniti - Integrazione con AirTable/Zapier per ricevere dati dei lead

### Architettura

- **Backend:** Hono framework su Cloudflare Workers
- **Frontend:** HTML/CSS/JavaScript con Tailwind CSS

- **Storage:** Cloudflare R2 per file allegati
- **Database:** Cloudflare D1 per configurazioni e template
- **API Esterne:** Google Calendar API, Gmail API

## Flusso Dati

AirTable/Zapier → Webhook → Cloudflare Worker → Google APIs  
→ Cloudflare R2  
→ Cloudflare D1

---

## □ FUNZIONALITÀ DA IMPLEMENTARE

---

### FUNZIONE 1: MODIFICA EVENTI GOOGLE CALENDAR

#### Descrizione

Ricevere dati da Zapier/AirTable e aggiornare automaticamente la descrizione di un evento Google Calendar esistente inserendo le informazioni del lead.

#### Requisiti Tecnici

##### 1.1 Setup Google Cloud

STEP 1: Creare progetto Google Cloud

- Vai su <https://console.cloud.google.com>
- Crea nuovo progetto: "TESTmess-Calendar"
- Abilita Google Calendar API

STEP 2: Creare credenziali OAuth2

- Tipo: OAuth 2.0 Client ID
- Tipo applicazione: Web application
- Authorized redirect URIs:  
<https://webapp.pages.dev/auth/google/callback>
- Scarica JSON credenziali

STEP 3: Configurare scopes necessari

- <https://www.googleapis.com/auth/calendar.events>
- <https://www.googleapis.com/auth/calendar>

##### 1.2 Configurazione Cloudflare

# *Salvare credenziali come secrets*

```
npx wrangler secret put GOOGLE_CLIENT_ID
npx wrangler secret put GOOGLE_CLIENT_SECRET
npx wrangler secret put GOOGLE_REFRESH_TOKEN
```

```
# Aggiungere a wrangler.jsonc
{
  "vars": {
    "GOOGLE_CALENDAR_ID": "primary"
  }
}
```

### 1.3 Endpoint API da Creare

POST /api/calendar/update-event

```
Body: {
  "eventId": "string",          // ID evento Google Calendar
  "leadData": {
    "nome": "string",
    "telefono": "string",
    "email": "string",
    "note": "string",
    "fonte": "string"
  }
}
```

```
Response: {
  "success": true,
  "eventUrl": "https://calendar.google.com/...",
  "updated": "2026-01-06T10:30:00Z"
}
```

### 1.4 Logica Implementazione

```
// src/routes/calendar.ts
```

1. Ricevere webhook da Zapier
2. Validare token di autenticazione
3. Estrarre eventId e leadData
4. Autenticare con Google Calendar API usando refresh token
5. Recuperare evento esistente
6. Formattare descrizione con dati lead:

LEAD INFORMATION

-----

Nome: {nome}

Telefono: {telefono}

Email: {email}

Fonte: {fonte}

```
Note:
{note}
```

```
[Descrizione originale evento]
```

7. Aggiornare evento su Google Calendar
8. Restituire conferma con link evento

## 1.5 Gestione Errori

- Token scaduto → Refresh automatico
- Evento non trovato → Errore 404
- Permission denied → Errore 403 con istruzioni
- Rate limit → Retry con backoff esponenziale

## 1.6 Testing

```
# Test locale
curl -X POST http://localhost:3000/api/calendar/update-event \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer TEST_TOKEN" \
  -d '{
    "eventId": "test123",
    "leadData": {
      "nome": "Mario Rossi",
      "telefono": "+39 333 1234567",
      "email": "mario.rossi@email.com",
      "note": "Cliente interessato a prodotto X",
      "fonte": "Facebook Ads"
    }
  }'
```

## 1.7 Configurazione Zapier

Trigger: New Record in AirTable (tabella "Leads")

Action: Webhooks by Zapier

- Method: POST
- URL: `https://webapp.pages.dev/api/calendar/update-event`
- Headers:
  - Authorization: Bearer {WEBHOOK\_SECRET}
- Body:

```
eventId: {Calendar Event ID}
leadData: {
  nome: {Name}
  telefono: {Phone}
  email: {Email}
  note: {Notes}
```

```
    fonte: {Source}
  }
```

## Criteri di Completamento

- ☐ Endpoint /api/calendar/update-event funzionante
  - ☐ Autenticazione Google OAuth2 configurata
  - ☐ Test con evento reale completato con successo
  - ☐ Gestione errori implementata
  - ☐ Documentazione API aggiornata
- 

# FUNZIONE 2: CREAZIONE BOZZE EMAIL AUTOMATICHE

## Descrizione

Ricevere dati da AirTable/Zapier e creare automaticamente una bozza email su Gmail con destinatario, oggetto e testo già compilati.

## Requisiti Tecnici

### 2.1 Setup Google Cloud (aggiuntivo a Funzione 1)

STEP 1: Abilitare Gmail API

- Stesso progetto "TESTmess-Calendar"
- Abilita Gmail API

STEP 2: Aggiungere scopes OAuth2

- <https://www.googleapis.com/auth/gmail.compose>
- <https://www.googleapis.com/auth/gmail.modify>
- Rifare autorizzazione OAuth2

### 2.2 Database Schema (Cloudflare D1)

```
-- Tabella per template email
CREATE TABLE email_templates (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  nome TEXT NOT NULL UNIQUE,
  tipo TEXT NOT NULL, -- 'preventivo', 'followup', 'brochure', etc.
  oggetto TEXT NOT NULL,
  corpo TEXT NOT NULL, -- Supporta placeholders: {{nome}}, {{telefono}}, etc.
  allegato_default_id INTEGER,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (allegato_default_id) REFERENCES allegati(id)
```

```
);

-- Indici
CREATE INDEX idx_email_templates_tipo ON email_templates(tipo);

-- Dati esempio
INSERT INTO email_templates (nome, tipo, oggetto, corpo) VALUES
('Preventivo Standard', 'preventivo',
 'Preventivo per {{nome}}',
 'Gentile {{nome}},\n\nGrazie per l''interesse mostrato. In allegato
   trova il preventivo richiesto.\n\nResto a
   disposizione.\n\nCordiali saluti'),
('Follow-up Chiamata', 'followup',
 'Follow-up conversazione del {{data}}',
 'Ciao {{nome}},\n\nTi scrivo in seguito alla nostra conversazione
   telefonica.\n\n{{note}}\n\nFammi sapere se hai
   domande.\n\nGrazie!');
```

## 2.3 Endpoint API da Creare

POST /api/email/create-draft

```
Body: {
  "destinatario": "email@example.com",
  "templateId": 1, // Opzionale, se vuoi usare template
  "oggetto": "string", // Opzionale se usi template
  "corpo": "string", // Opzionale se usi template
  "datilead": { // Per sostituire placeholders
    "nome": "string",
    "telefono": "string",
    "data": "string",
    "note": "string"
  },
  "allegati": [ // Array di ID allegati (vedi Funzione 4)
    1, 3, 5
  ]
}
```

```
Response: {
  "success": true,
  "draftId": "r-1234567890",
  "draftUrl": "https://mail.google.com/mail/u/0/#drafts?compose=r-1234567890",
  "preview": {
    "to": "email@example.com",
    "subject": "Preventivo per Mario Rossi",
    "bodyPreview": "Gentile Mario Rossi, Grazie per..."
  }
}
```

## 2.4 Logica Implementazione

```
// src/routes/email.ts
```

1. Ricevere richiesta con dati email
2. Se templateId fornito:
  - a. **Recuperare** template da D1
  - b. **Sostituire** placeholders con datiLead
  - c. **Recuperare** allegati **default**
3. Se oggetto/corpo forniti direttamente, usare quelli
4. Autenticare con Gmail API
5. Creare bozza email:
  - Codificare corpo **in** base64url
  - Formattare headers MIME
  - Allegare file se **presenti** (vedi Funzione 3)
6. Salvare bozza su Gmail
7. Restituire link bozza

## 2.5 Funzione Sostituzione Placeholders

```
function replacePlaceholders(text: string, data: any): string {  
  return text.replace(/\{\{(\w+)\}\}/g, (match, key) => {  
    return data[key] || match;  
  });  
}
```

```
// Esempio:  
// Input: "Ciao {{nome}}, il tuo numero è {{telefono}}"  
// Data: {nome: "Mario", telefono: "333-1234567"}  
// Output: "Ciao Mario, il tuo numero è 333-1234567"
```

## 2.6 Formato Email MIME

```
const emailContent = [  
  'Content-Type: text/plain; charset=utf-8',  
  'MIME-Version: 1.0',  
  `To: ${destinatario}`,  
  `Subject: ${oggetto}`,  
  '',  
  corpo  
].join('\n');
```

```
const encodedEmail = btoa(emailContent)  
  .replace(/\+/g, '-')  
  .replace(/\//g, '_')  
  .replace(/=+$/ , '');
```

## 2.7 Testing

```
# Test creazione bozza semplice
curl -X POST http://localhost:3000/api/email/create-draft \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer TEST_TOKEN" \
  -d '{
    "destinatario": "test@example.com",
    "oggetto": "Test Email",
    "corpo": "Questo è un test"
  }'
```

```
# Test con template
curl -X POST http://localhost:3000/api/email/create-draft \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer TEST_TOKEN" \
  -d '{
    "destinatario": "mario.rossi@email.com",
    "templateId": 1,
    "datiLead": {
      "nome": "Mario Rossi",
      "telefono": "+39 333 1234567",
      "data": "06/01/2026"
    }
  }'
```

## 2.8 UI per Gestione Template

```
<!-- Pagina /templates -->
<div class="template-manager">
  <h2>Template Email</h2>

  <button id="new-template">+ Nuovo Template</button>

  <div class="template-list">
    <!-- Lista template con edit/delete -->
  </div>

  <div class="template-editor" style="display:none">
    <input type="text" name="nome" placeholder="Nome template">
    <select name="tipo">
      <option value="preventivo">Preventivo</option>
      <option value="followup">Follow-up</option>
      <option value="brochure">Brochure</option>
    </select>
    <input type="text" name="oggetto" placeholder="Oggetto (usa {{nome}}, {{telefono}}, etc.)">
    <textarea name="corpo" placeholder="Corpo email (usa placeholders)"></textarea>
    <select name="allegato_default">
      <option value="">Nessun allegato</option>
```

```
        <!-- Lista allegati da Funzione 4 -->
    </select>
    <button class="save">Salva</button>
    <button class="cancel">Annulla</button>
</div>
</div>
```

## Criteri di Completamento

- ☐ Endpoint /api/email/create-draft funzionante
  - ☐ Sistema template con placeholders implementato
  - ☐ Database D1 con tabella email\_templates
  - ☐ UI per gestione template
  - ☐ Test con Gmail reale completato
  - ☐ Integrazione con Zapier configurata
- 

# FUNZIONE 3: GESTIONE ALLEGATI EMAIL

## Descrizione

Permettere di allegare file alle bozze email create automaticamente, con supporto per file salvati su cloud storage (Cloudflare R2 o Google Drive).

## Requisiti Tecnici

### 3.1 Scelta Architettura Storage

**OPZIONE A: Cloudflare R2 (CONSIGLIATA)** - ☐ Integrato con Cloudflare Workers - ☐ Nessun costo per storage (primi 10GB gratis) - ☐ Nessun costo per download (primo 1 milione gratis al mese) - ☐ API semplice - ⚠ Limite 10MB per request su Workers

**OPZIONE B: Google Drive** - ☐ Integrato con Google Workspace - ☐ 15GB gratis - ⚠ Richiede OAuth2 aggiuntivo - ⚠ API più complessa

**SCELTA: Cloudflare R2** per semplicità e costi

### 3.2 Configurazione Cloudflare R2

```
# Creare bucket R2
npx wrangler r2 bucket create testmess-attachments

# Aggiungere a wrangler.jsonc
{
```

```

    "r2_buckets": [
      {
        "binding": "ATTACHMENTS",
        "bucket_name": "testmess-attachments"
      }
    ]
  }
}

```

### 3.3 Database Schema (aggiunta a D1)

```

-- Tabella allegati
CREATE TABLE allegati (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  nome_file TEXT NOT NULL,
  nome_originale TEXT NOT NULL,
  tipo_mime TEXT NOT NULL,
  dimensione INTEGER NOT NULL, -- in bytes
  r2_key TEXT NOT NULL UNIQUE, -- Percorso in R2
  categoria TEXT, -- 'preventivo', 'brochure', 'contratto', etc.
  descrizione TEXT,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Indici
CREATE INDEX idx_allegati_categoria ON allegati(categoria);
CREATE INDEX idx_allegati_r2_key ON allegati(r2_key);

-- Tabella associazione email-allegati (per tracking)
CREATE TABLE email_allegati (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  draft_id TEXT NOT NULL,
  allegato_id INTEGER NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (allegato_id) REFERENCES allegati(id)
);

```

### 3.4 Endpoint API da Creare

#### Upload Allegato:

POST /api/attachments/upload  
 Content-Type: multipart/form-data

Body:

```

file: [binary]
categoria: string (opzionale)
descrizione: string (opzionale)

```

```
Response: {
  "success": true,
  "attachment": {
    "id": 1,
    "nome_file": "preventivo_2026.pdf",
    "r2_key": "attachments/2026-01-06/abc123.pdf",
    "url": "https://testmess-attachments.r2.dev/attachments/2026-01-06/abc123.pdf",
    "dimensione": 245678,
    "tipo_mime": "application/pdf"
  }
}
```

### **Elenco Allegati:**

GET /api/attachments?categoria=preventivo

```
Response: {
  "success": true,
  "attachments": [
    {
      "id": 1,
      "nome_originale": "Preventivo Standard 2026.pdf",
      "categoria": "preventivo",
      "dimensione": 245678,
      "created_at": "2026-01-06T10:00:00Z"
    }
  ]
}
```

### **Download Allegato:**

GET /api/attachments/:id/download

Response: [binary file]

Headers:

Content-Type: application/pdf

Content-Disposition: attachment; filename="preventivo.pdf"

### **Eliminazione Allegato:**

DELETE /api/attachments/:id

```
Response: {
  "success": true,
  "message": "Allegato eliminato"
}
```

## **3.5 Logica Upload File**

```

// src/routes/attachments.ts

async function handleUpload(c: Context) {
  // 1. Ricevere file da form multipart
  const formData = await c.req.formData();
  const file = formData.get('file') as File;

  // 2. Validare file
  if (!file) throw new Error('File mancante');
  if (file.size > 10 * 1024 * 1024) {
    throw new Error('File troppo grande (max 10MB)');
  }

  // 3. Generare chiave unica R2
  const date = new Date().toISOString().split('T')[0];
  const randomId = crypto.randomUUID();
  const ext = file.name.split('.').pop();
  const r2Key = `attachments/${date}/${randomId}.${ext}`;

  // 4. Salvare su R2
  const buffer = await file.arrayBuffer();
  await c.env.ATTACHMENTS.put(r2Key, buffer, {
    httpMetadata: {
      contentType: file.type
    }
  });

  // 5. Salvare metadati su D1
  const result = await c.env.DB.prepare(`
    INSERT INTO allegati
    (nome_file, nome_originale, tipo_mime, dimensione, r2_key,
     categoria, descrizione)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?)
  `).bind(
    file.name,
    file.name,
    file.type,
    file.size,
    r2Key,
    formData.get('categoria') || null,
    formData.get('descrizione') || null
  ).run();

  return c.json({
    success: true,
    attachment: {
      id: result.meta.last_row_id,
      nome_file: file.name,

```

```

        r2_key: r2Key,
        dimensione: file.size,
        tipo_mime: file.type
    }
});
}

```

### 3.6 Integrazione con Gmail API

*// Modificare funzione create-draft per supportare allegati*

```

async function createDraftWithAttachments(
    destinatario: string,
    oggetto: string,
    corpo: string,
    allegatoIds: number[]
) {
    // 1. Recuperare allegati da D1
    const allegati = await getAllegatiByIds(allegatoIds);

    // 2. Scaricare file da R2
    const fileBuffers = await Promise.all(
        allegati.map(async (a) => {
            const obj = await c.env.ATTACHMENTS.get(a.r2_key);
            return {
                filename: a.nome_originale,
                mimeType: a.tipo_mime,
                data: await obj.arrayBuffer()
            };
        })
    );

    // 3. Costruire email MIME multipart
    const boundary = '----boundary' + Date.now();

    const parts = [
        // Parte 1: testo email
        `--${boundary}`,
        'Content-Type: text/plain; charset=utf-8',
        '',
        corpo,
        ''
    ];

    // Parte 2+: allegati
    for (const file of fileBuffers) {
        parts.push(
            `--${boundary}`,

```

```

        `Content-Type: ${file.mimeType}`,
        'Content-Transfer-Encoding: base64',
        `Content-Disposition: attachment;
          filename="${file.filename}"`,
        '',
        Buffer.from(file.data).toString('base64'),
        ''
      );
    };
  }

  parts.push(`--${boundary}--`);

  const email = [
    `To: ${destinatario}`,
    `Subject: ${oggetto}`,
    `Content-Type: multipart/mixed; boundary="${boundary}"`,
    '',
    ...parts
  ].join('\r\n');

  // 4. Codificare e inviare a Gmail
  const encodedEmail = btoa(email)
    .replace(/\+/g, '-')
    .replace(/\//g, '_')
    .replace(/=+$/, '');

  const response = await fetch(
    'https://www.googleapis.com/gmail/v1/users/me/drafts',
    {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${accessToken}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        message: { raw: encodedEmail }
      })
    }
  );

  return response.json();
}

```

### 3.7 UI Upload Allegati

```

<!-- Pagina /attachments -->
<div class="attachments-manager">
  <h2>Gestione Allegati</h2>

```

```

<div class="upload-section">
  <form id="upload-form" enctype="multipart/form-data">
    <input type="file" name="file" required>
    <select name="categoria">
      <option value="">Seleziona categoria</option>
      <option value="preventivo">Preventivo</option>
      <option value="brochure">Brochure</option>
      <option value="contratto">Contratto</option>
      <option value="altro">Altro</option>
    </select>
    <input type="text" name="descrizione" placeholder="Descrizione
      (opzionale)">
    <button type="submit">Carica</button>
  </form>
  <div class="upload-progress" style="display:none">
    <progress value="0" max="100"></progress>
    <span class="progress-text">0%</span>
  </div>
</div>

<div class="attachments-list">
  <h3>Allegati Caricati</h3>
  <div class="filters">
    <select id="filter-categoria">
      <option value="">Tutte le categorie</option>
      <option value="preventivo">Preventivo</option>
      <option value="brochure">Brochure</option>
      <option value="contratto">Contratto</option>
    </select>
  </div>

  <table>
    <thead>
      <tr>
        <th>Nome File</th>
        <th>Categoria</th>
        <th>Dimensione</th>
        <th>Data Upload</th>
        <th>Azioni</th>
      </tr>
    </thead>
    <tbody id="attachments-tbody">
      <!-- Popolato dinamicamente -->
    </tbody>
  </table>
</div>
</div>

```

```

<script>
  // Gestione upload con progress bar
  document.getElementById('upload-form').addEventListener('submit',
    async (e) => {
      e.preventDefault();

      const formData = new FormData(e.target);
      const progressBar = document.querySelector('.upload-progress');
      const progress = progressBar.querySelector('progress');
      const progressText = progressBar.querySelector('.progress-text');

      progressBar.style.display = 'block';

      try {
        const xhr = new XMLHttpRequest();

        xhr.upload.addEventListener('progress', (e) => {
          if (e.lengthComputable) {
            const percent = (e.loaded / e.total) * 100;
            progress.value = percent;
            progressText.textContent = Math.round(percent) + '%';
          }
        });

        xhr.addEventListener('load', () => {
          if (xhr.status === 200) {
            alert('File caricato con successo!');
            location.reload();
          } else {
            alert('Errore durante il caricamento');
          }
          progressBar.style.display = 'none';
        });

        xhr.open('POST', '/api/attachments/upload');
        xhr.send(formData);

      } catch (error) {
        console.error(error);
        alert('Errore durante il caricamento');
        progressBar.style.display = 'none';
      }
    });
</script>

```

### 3.8 Limitazioni e Workaround

## Problema: Limite 10MB su Cloudflare Workers

### Soluzione A: Upload diretto a R2 (CONSIGLIATA)

```
// 1. Frontend richiede signed URL
const response = await fetch('/api/attachments/upload-url', {
  method: 'POST',
  body: JSON.stringify({
    filename: file.name,
    contentType: file.type
  })
});
const { uploadUrl, r2Key } = await response.json();

// 2. Upload diretto a R2 (bypassa Worker)
await fetch(uploadUrl, {
  method: 'PUT',
  body: file,
  headers: { 'Content-Type': file.type }
});

// 3. Notifica Worker che upload è completato
await fetch('/api/attachments/confirm', {
  method: 'POST',
  body: JSON.stringify({ r2Key, filename: file.name, size: file.size })
});
```

### Soluzione B: Chunked upload

```
// Upload file in chunk da 5MB
const chunkSize = 5 * 1024 * 1024;
const chunks = Math.ceil(file.size / chunkSize);

for (let i = 0; i < chunks; i++) {
  const start = i * chunkSize;
  const end = Math.min(start + chunkSize, file.size);
  const chunk = file.slice(start, end);

  await fetch('/api/attachments/upload-chunk', {
    method: 'POST',
    headers: {
      'X-Chunk-Index': i.toString(),
      'X-Total-Chunks': chunks.toString(),
      'X-Upload-Id': uploadId
    },
    body: chunk
  });
}
```

## Criteri di Completamento

- ☐ Cloudflare R2 configurato
  - ☐ Endpoint upload/download/delete funzionanti
  - ☐ Database D1 con tabella allegati
  - ☐ UI per gestione allegati
  - ☐ Integrazione con Gmail API per allegati email
  - ☐ Test upload file < 10MB
  - ☐ Test upload file > 10MB (con signed URL)
  - ☐ Test creazione bozza con allegati
- 

## FUNZIONE 4: SEZIONE ALLEGATI CON TEMPLATE DEFAULT

### Descrizione

Creare un sistema di gestione allegati con associazione automatica ai template email, permettendo di definire quale allegato viene automaticamente incluso per ogni tipo di messaggio.

### Requisiti Tecnici

#### 4.1 Database Schema (già creato in Funzione 2 e 3)

```
-- Già creato in Funzione 2:
-- email_templates.allegato_default_id → FK a allegati.id

-- Già creato in Funzione 3:
-- allegati.categoria

-- Nuova tabella: associazioni multiple template-allegati
CREATE TABLE template_allegati (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  template_id INTEGER NOT NULL,
  allegato_id INTEGER NOT NULL,
  ordine INTEGER DEFAULT 1, -- Per ordinare allegati
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (template_id) REFERENCES email_templates(id) ON DELETE CASCADE,
  FOREIGN KEY (allegato_id) REFERENCES allegati(id) ON DELETE CASCADE,
  UNIQUE(template_id, allegato_id)
);

CREATE INDEX idx_template_allegati_template ON
  template_allegati(template_id);
```

```
CREATE INDEX idx_template_allegati_allegato ON
  template_allegati(allegato_id);
```

## 4.2 Logica Template con Allegati Default

### Caso 1: Allegato singolo default (allegato\_default\_id)

```
// Quando si crea bozza da template con allegato_default_id
const template = await db.prepare(
  'SELECT * FROM email_templates WHERE id = ?'
).bind(templateId).first();

if (template.allegato_default_id) {
  allegatoIds.push(template.allegato_default_id);
}
```

### Caso 2: Allegati multipli (template\_allegati)

```
// Quando si crea bozza da template con allegati multipli
const allegati = await db.prepare(`
  SELECT a.*
  FROM allegati a
  JOIN template_allegati ta ON a.id = ta.allegato_id
  WHERE ta.template_id = ?
  ORDER BY ta.ordine
`).bind(templateId).all();

allegatoIds.push(...allegati.results.map(a => a.id));
```

## 4.3 Endpoint API da Creare

### Associa allegati a template:

POST /api/templates/:templateId/attachments

Body: {  
 "allegatoIds": [1, 3, 5] // Array di ID allegati da associare  
}

Response: {  
 "success": true,  
 "template": {  
 "id": 1,  
 "nome": "Preventivo Standard",  
 "allegati": [  
 { "id": 1, "nome": "Preventivo\_template.pdf" },  
 { "id": 3, "nome": "Condizioni\_vendita.pdf" },  
 { "id": 5, "nome": "Brochure\_prodotti.pdf" }  
 ]  
 }  
}

### Recupera allegati di un template:

GET /api/templates/:templateId/attachments

```
Response: {
  "success": true,
  "allegati": [
    {
      "id": 1,
      "nome_originale": "Preventivo_template.pdf",
      "dimensione": 245678,
      "ordine": 1
    },
    {
      "id": 3,
      "nome_originale": "Condizioni_vendita.pdf",
      "dimensione": 128456,
      "ordine": 2
    }
  ]
}
```

### Riordina allegati template:

PUT /api/templates/:templateId/attachments/reorder

```
Body: {
  "ordine": [3, 1, 5] // Nuovo ordine ID allegati
}
```

```
Response: {
  "success": true
}
```

### Rimuovi allegato da template:

DELETE /api/templates/:templateId/attachments/:allegatoId

```
Response: {
  "success": true
}
```

## 4.4 UI Gestione Template con Allegati

```
<!-- Estensione della UI template da Funzione 2 -->
<div class="template-editor">
  <input type="text" name="nome" placeholder="Nome template">
  <select name="tipo">
    <option value="preventivo">Preventivo</option>
    <option value="followup">Follow-up</option>
```

```

        <option value="brochure">Brochure</option>
    </select>
    <input type="text" name="oggetto" placeholder="Oggetto">
    <textarea name="corpo" placeholder="Corpo email"></textarea>

    <!-- NUOVA SEZIONE ALLEGATI -->
    <div class="template-attachments">
        <h4>Allegati Default</h4>
        <p class="help-text">
            Questi allegati verranno inclusi automaticamente
            quando usi questo template
        </p>

        <div class="attachments-selector">
            <button type="button" id="add-attachment">
                + Aggiungi Allegato
            </button>

            <div class="selected-attachments" id="selected-attachments">
                <!-- Lista allegati selezionati (drag & drop per riordinare)
                -->
            </div>
        </div>
    </div>

    <button class="save">Salva Template</button>
    <button class="cancel">Annulla</button>
</div>

<!-- Modal selezione allegati -->
<div class="modal" id="attachment-modal" style="display:none">
    <div class="modal-content">
        <h3>Seleziona Allegati</h3>

        <div class="search-bar">
            <input type="text" id="search-attachments"
                placeholder="Cerca per nome...">
            <select id="filter-categoria">
                <option value="">Tutte le categorie</option>
                <option value="preventivo">Preventivo</option>
                <option value="brochure">Brochure</option>
                <option value="contratto">Contratto</option>
            </select>
        </div>

        <div class="attachments-grid">
            <!-- Griglia allegati con checkbox -->
            <div class="attachment-card" data-id="1">

```

```

        <input type="checkbox" value="1">
        <i class="fas fa-file-pdf"></i>
        <span class="filename">Preventivo_template.pdf</span>
        <span class="filesize">240 KB</span>
    </div>
    <!-- ... altri allegati ... -->
</div>

<div class="modal-actions">
    <button class="confirm">Conferma Selezione</button>
    <button class="cancel">Annulla</button>
</div>
</div>
</div>

<script>
// Drag & drop per riordinare allegati
const selectedAttachments = document.getElementById('selected-
    attachments');
let draggedElement = null;

selectedAttachments.addEventListener('dragstart', (e) => {
    draggedElement = e.target;
    e.target.style.opacity = '0.5';
});

selectedAttachments.addEventListener('dragend', (e) => {
    e.target.style.opacity = '1';
});

selectedAttachments.addEventListener('dragover', (e) => {
    e.preventDefault();
    const afterElement = getDragAfterElement(selectedAttachments,
        e.clientY);
    if (afterElement == null) {
        selectedAttachments.appendChild(draggedElement);
    } else {
        selectedAttachments.insertBefore(draggedElement, afterElement);
    }
});

function getDragAfterElement(container, y) {
    const draggableElements = [
        ...container.querySelectorAll('.attachment-item:not(.dragging)')
    ];

    return draggableElements.reduce((closest, child) => {
        const box = child.getBoundingClientRect();
        const offset = y - box.top - box.height / 2;

```

```

        if (offset < 0 && offset > closest.offset) {
            return { offset: offset, element: child };
        } else {
            return closest;
        }
    }, { offset: Number.NEGATIVE_INFINITY }).element;
}

// Salvataggio template con allegati
document.querySelector('.template-editor
    .save').addEventListener('click', async () => {
    const templateData = {
        nome: document.querySelector('[name="nome"]').value,
        tipo: document.querySelector('[name="tipo"]').value,
        oggetto: document.querySelector('[name="oggetto"]').value,
        corpo: document.querySelector('[name="corpo"]').value
    };

    // Salva template
    const response = await fetch('/api/templates', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(templateData)
    });
    const { templateId } = await response.json();

    // Salva associazioni allegati
    const allegatoIds =
        [...selectedAttachments.querySelectorAll('.attachment-
            item')]
        .map(item => parseInt(item.dataset.id));

    if (allegatoIds.length > 0) {
        await fetch(`/api/templates/${templateId}/attachments`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ allegatoIds })
        });
    }

    alert('Template salvato con successo!');
    location.reload();
});
</script>

```

## 4.5 Dashboard Riepilogativa

```

<!-- Pagina /dashboard -->

```

```

<div class="dashboard">
  <h1>Dashboard TESTmess</h1>

  <div class="stats-cards">
    <div class="stat-card">
      <i class="fas fa-envelope"></i>
      <h3>Template Email</h3>
      <div class="number" id="total-templates">0</div>
    </div>

    <div class="stat-card">
      <i class="fas fa-paperclip"></i>
      <h3>Allegati Totali</h3>
      <div class="number" id="total-attachments">0</div>
    </div>

    <div class="stat-card">
      <i class="fas fa-calendar"></i>
      <h3>Eventi Aggiornati</h3>
      <div class="number" id="total-events">0</div>
      <small>Ultimi 30 giorni</small>
    </div>

    <div class="stat-card">
      <i class="fas fa-envelope-open-text"></i>
      <h3>Bozze Create</h3>
      <div class="number" id="total-drafts">0</div>
      <small>Ultimi 30 giorni</small>
    </div>
  </div>

  <div class="templates-overview">
    <h2>Template Email</h2>
    <table>
      <thead>
        <tr>
          <th>Nome</th>
          <th>Tipo</th>
          <th>Allegati</th>
          <th>Ultimo Utilizzo</th>
          <th>Azioni</th>
        </tr>
      </thead>
      <tbody id="templates-tbody">
        <!-- Popolato dinamicamente -->
      </tbody>
    </table>
  </div>

```

```

<div class="quick-actions">
  <h2>Azioni Rapide</h2>
  <div class="actions-grid">
    <a href="/templates" class="action-button">
      <i class="fas fa-plus"></i>
      Nuovo Template
    </a>
    <a href="/attachments" class="action-button">
      <i class="fas fa-upload"></i>
      Carica Allegato
    </a>
    <a href="/settings" class="action-button">
      <i class="fas fa-cog"></i>
      Impostazioni
    </a>
  </div>
</div>
</div>

```

## 4.6 Logica Creazione Bozza Completa

```

// src/routes/email.ts - Versione finale con tutto

async function createDraft(c: Context) {
  const { destinatario, templateId, oggetto, corpo, datilead,
    allegatoIds } =
    await c.req.json();

  let finalOggetto = oggetto;
  let finalCorpo = corpo;
  let finalAllegati = allegatoIds || [];

  // Se template fornito, usare quello
  if (templateId) {
    const template = await c.env.DB.prepare(
      'SELECT * FROM email_templates WHERE id = ?'
    ).bind(templateId).first();

    if (!template) {
      return c.json({ error: 'Template non trovato' }, 404);
    }

    // Sostituire placeholders
    finalOggetto = replacePlaceholders(template.oggetto, datilead);
    finalCorpo = replacePlaceholders(template.corpo, datilead);

    // Recuperare allegati default del template

```

```

const templateAllegati = await c.env.DB.prepare(`
  SELECT allegato_id
  FROM template_allegati
  WHERE template_id = ?
  ORDER BY ordine
`).bind(templateId).all();

finalAllegati = templateAllegati.results.map(r =>
  r.allegato_id);
}

// Creare bozza Gmail con allegati
const draft = await createGmailDraft(
  c,
  destinatario,
  finalOggetto,
  finalCorpo,
  finalAllegati
);

// Salvare tracking
await c.env.DB.prepare(`
  INSERT INTO email_allegati (draft_id, allegato_id)
  VALUES ${finalAllegati.map(() => '(?, ?)').join(', ')}
`).bind(...finalAllegati.flatMap(id => [draft.id, id])).run();

return c.json({
  success: true,
  draftId: draft.id,
  draftUrl: `https://mail.google.com/mail/u/0/#drafts?compose=${draft.id}`,
  preview: {
    to: destinatario,
    subject: finalOggetto,
    bodyPreview: finalCorpo.substring(0, 100) + '...',
    attachments: finalAllegati.length
  }
});
}

```

## Criteri di Completamento

- ☐ Tabella template\_allegati creata in D1
  - ☐ Endpoint associazione template-allegati funzionanti
  - ☐ UI drag & drop per riordinare allegati
  - ☐ Dashboard riepilogativa funzionante
  - ☐ Logica completa creazione bozza con template e allegati
  - ☐ Test end-to-end: template → bozza email con allegati
-

## □ TESTING COMPLETO

### Test Funzione 1: Google Calendar

*# 1. Test autenticazione*

```
curl http://localhost:3000/api/calendar/auth/status
```

*# 2. Test update evento*

```
curl -X POST http://localhost:3000/api/calendar/update-event \
-H "Content-Type: application/json" \
-d '{
  "eventId": "YOUR_TEST_EVENT_ID",
  "leadData": {
    "nome": "Mario Rossi Test",
    "telefono": "+39 333 1234567",
    "email": "test@example.com",
    "note": "Lead di test",
    "fonte": "Test Manuale"
  }
}'
```

*# 3. Verificare su Google Calendar che la descrizione sia stata aggiornata*

### Test Funzione 2: Gmail Bozze

*# 1. Test creazione bozza semplice*

```
curl -X POST http://localhost:3000/api/email/create-draft \
-H "Content-Type: application/json" \
-d '{
  "destinatario": "test@example.com",
  "oggetto": "Test Bozza",
  "corpo": "Questo è un test"
}'
```

*# 2. Test creazione bozza da template*

```
curl -X POST http://localhost:3000/api/email/create-draft \
-H "Content-Type: application/json" \
-d '{
  "destinatario": "test@example.com",
  "templateId": 1,
  "datiLead": {
    "nome": "Mario Rossi",
    "telefono": "+39 333 1234567"
  }
}'
```

*# 3. Verificare su Gmail che la bozza sia stata creata*

## Test Funzione 3: Allegati

*# 1. Test upload allegato*

```
curl -X POST http://localhost:3000/api/attachments/upload \
  -F "file=@/path/to/test.pdf" \
  -F "categoria=preventivo" \
  -F "descrizione=Test allegato"
```

*# 2. Test lista allegati*

```
curl http://localhost:3000/api/attachments
```

*# 3. Test download allegato*

```
curl http://localhost:3000/api/attachments/1/download -o
downloaded.pdf
```

*# 4. Test bozza con allegati*

```
curl -X POST http://localhost:3000/api/email/create-draft \
  -H "Content-Type: application/json" \
  -d '{
    "destinatario": "test@example.com",
    "oggetto": "Test con allegato",
    "corpo": "Email con PDF allegato",
    "allegatoIds": [1]
  }'
```

## Test Funzione 4: Template con Allegati

*# 1. Creare template*

```
curl -X POST http://localhost:3000/api/templates \
  -H "Content-Type: application/json" \
  -d '{
    "nome": "Preventivo Test",
    "tipo": "preventivo",
    "oggetto": "Preventivo per {{nome}}",
    "corpo": "Gentile {{nome}},\n\nIn allegato il preventivo."
  }'
```

*# 2. Associare allegati al template*

```
curl -X POST http://localhost:3000/api/templates/1/attachments \
  -H "Content-Type: application/json" \
  -d '{
    "allegatoIds": [1, 2]
  }'
```

*# 3. Creare bozza da template (deve includere allegati automaticamente)*

```
curl -X POST http://localhost:3000/api/email/create-draft \
  -H "Content-Type: application/json" \
  -d '{
```

```
"destinatario": "test@example.com",
"templateId": 1,
"datilead": {
  "nome": "Mario Rossi"
}
}'
```

*# 4. Verificare che la bozza su Gmail abbia 2 allegati*

---

## ☐ DEPLOYMENT CHECKLIST

### Pre-Deployment

- ☐ Tutte le funzioni testate localmente
- ☐ Database D1 migrations applicate in locale
- ☐ Tutti i secrets configurati localmente (.dev.vars)
- ☐ Build progetto completata senza errori
- ☐ README.md aggiornato
- ☐ Git commit con tutte le modifiche

### Cloudflare Setup

- ☐ Database D1 production creato
- ☐ Bucket R2 production creato
- ☐ Migrations applicate su production D1
- ☐ Secrets configurati su production:

```
npx wrangler secret put GOOGLE_CLIENT_ID
npx wrangler secret put GOOGLE_CLIENT_SECRET
npx wrangler secret put GOOGLE_REFRESH_TOKEN
npx wrangler secret put WEBHOOK_SECRET
```

- ☐ wrangler.jsonc configurato correttamente
- ☐ Build production completata

### Deployment

*# 1. Build*

```
cd /home/user/webapp && npm run build
```

*# 2. Apply migrations production*

```
npx wrangler d1 migrations apply webapp-production
```

*# 3. Deploy*

```
npx wrangler pages deploy dist --project-name webapp
```

*# 4. Verificare URL production*

```
curl https://webapp.pages.dev/api/health
```

## Post-Deployment

- ☐ Test Google Calendar update su production
- ☐ Test Gmail draft creation su production
- ☐ Test upload allegati su production
- ☐ Test template con allegati su production
- ☐ Configurare Zapier con URL production
- ☐ Testare webhook Zapier → Cloudflare
- ☐ Aggiornare cloudflare\_project\_name in meta\_info
- ☐ Documentare URL production in README.md

## Zapier Configuration

Zap 1: AirTable Lead → Google Calendar

Trigger: New record in AirTable (table: Leads)

Action: Webhook POST

URL: `https://webapp.pages.dev/api/calendar/update-event`

Headers: Authorization: Bearer {WEBHOOK\_SECRET}

Body: {eventId, leadData}

Zap 2: AirTable Lead → Gmail Draft

Trigger: New record in AirTable (table: Leads)

Action: Webhook POST

URL: `https://webapp.pages.dev/api/email/create-draft`

Headers: Authorization: Bearer {WEBHOOK\_SECRET}

Body: {destinatario, templateId, datiLead}

---

## 📁 STRUTTURA FINALE PROGETTO

```
/home/user/webapp/
```

```
├─ docs/
```

```
|   └─ REQUIREMENTS_SPECIFICATIONS.md (questo file)
```

```
|   └─ API_DOCUMENTATION.md
```

```
├─ src/
```

```
|   └─ index.tsx (entry point Hono)
```

```
|   └─ routes/
```

```
|       └─ calendar.ts (Funzione 1)
```

```
|       └─ email.ts (Funzione 2)
```

```
|       └─ attachments.ts (Funzione 3)
```

```
|       └─ templates.ts (Funzione 4)
```

```
|   └─ services/
```

```

|   |   |─ google-auth.ts           (OAuth2 helper)
|   |   |─ google-calendar.ts       (Calendar API wrapper)
|   |   └─ gmail.ts                 (Gmail API wrapper)
|   └─ utils/
|       |─ placeholders.ts          (Template placeholders)
|       |─ mime.ts                  (Email MIME builder)
|       └─ validation.ts            (Input validation)
|   └─ types/
|       └─ index.ts                 (TypeScript types)
└─ migrations/
    |─ 0001_initial_schema.sql
    |─ 0002_email_templates.sql
    |─ 0003_allegati.sql
    └─ 0004_template_allegati.sql
└─ public/
    |─ static/
    |   |─ app.js                   (Frontend JavaScript)
    |   |─ styles.css               (Custom CSS)
    |   └─ favicon.ico
└─ .git/
└─ .gitignore
└─ .dev.vars                        (local secrets)
└─ ecosystem.config.cjs            (PM2 config)
└─ wrangler.jsonc                  (Cloudflare config)
└─ vite.config.ts                  (Vite config)
└─ package.json
└─ tsconfig.json
└─ README.md

```

---

## □ ENVIRONMENT VARIABLES

### Local Development (.dev.vars)

*# Google OAuth2*

GOOGLE\_CLIENT\_ID=your-client-id.apps.googleusercontent.com

GOOGLE\_CLIENT\_SECRET=your-client-secret

GOOGLE\_REFRESH\_TOKEN=your-refresh-token

*# Webhook Security*

WEBHOOK\_SECRET=generate-random-secure-string

*# Google Calendar*

GOOGLE\_CALENDAR\_ID=primary

### Production (Cloudflare Secrets)

*# Configurare con:*

```
npx wrangler secret put GOOGLE_CLIENT_ID
npx wrangler secret put GOOGLE_CLIENT_SECRET
npx wrangler secret put GOOGLE_REFRESH_TOKEN
npx wrangler secret put WEBHOOK_SECRET
```

---

## 📄 COME USARE QUESTO DOCUMENTO

### Per una nuova sessione AI:

1. Leggi questo file per intero
2. Vai alla sezione “STATO IMPLEMENTAZIONE”
3. Identifica cosa è completato e cosa manca
4. Chiedi all’utente quale funzione vuole implementare
5. Lavora su UNA funzione alla volta
6. Aggiorna “STATO IMPLEMENTAZIONE” al completamento
7. Fai commit git con messaggio descrittivo

### Per l’utente:

1. Carica questo file insieme alla cartella del progetto
2. Di all’AI: “Leggi il file REQUIREMENTS\_SPECIFICATIONS.md e dimmi lo stato del progetto”
3. Specifica quale funzione vuoi implementare
4. L’AI lavorerà seguendo le specifiche di questo documento

### Per aggiornamenti:

Quando aggiungi nuove funzionalità: 1. Aggiungi una nuova sezione “FUNZIONE X” 2. Segui lo stesso formato delle altre funzioni 3. Aggiorna “STATO IMPLEMENTAZIONE” 4. Aggiorna “DEPLOYMENT CHECKLIST” se necessario

---

## 🆘 TROUBLESHOOTING

### Errore: “Invalid credentials” Google API

*# 1. Verificare che i secrets siano configurati*

```
npx wrangler secret list
```

*# 2. Rigenerare refresh token*

*# Vai su: <https://developers.google.com/oauthplayground>*

*# Seleziona scopes: Calendar API v3 e Gmail API v1*

*# Autorizza e copia il refresh token*

```
# Aggiorna secret: npx wrangler secret put GOOGLE_REFRESH_TOKEN
```

## **Errore: “File too large” (> 10MB)**

```
// Usare signed URL per upload diretto a R2  
// Vedi sezione 3.8 "Limitazioni e Workaround"
```

## **Errore: “Template not found”**

```
# Verificare che il database abbia dati  
npx wrangler d1 execute webapp-production --local \  
  --command="SELECT * FROM email_templates"
```

```
# Se vuoto, inserire template di esempio  
npx wrangler d1 execute webapp-production --local \  
  --file=./seed_templates.sql
```

## **Webhook Zapier non funziona**

```
# 1. Testare endpoint manualmente  
curl -X POST https://webapp.pages.dev/api/calendar/update-event \  
  -H "Authorization: Bearer YOUR_WEBHOOK_SECRET" \  
  -H "Content-Type: application/json" \  
  -d '{"eventId":"test","leadData":{"nome":"Test"}}'
```

```
# 2. Verificare logs Cloudflare  
npx wrangler tail
```

```
# 3. Verificare che WEBHOOK_SECRET sia configurato  
npx wrangler secret list
```

---

# **☐ METRICHE E MONITORING**

## **Logging**

```
// Aggiungere logging a tutte le funzioni critiche  
console.log('[CALENDAR] Updating event:', eventId);  
console.log('[EMAIL] Creating draft for:', destinatario);  
console.log('[ATTACHMENTS] Uploading file:', filename);
```

```
// Cloudflare Workers: i log sono visibili con  
npx wrangler tail
```

## **Statistiche da Tracciare**

```
-- Creare tabella analytics
```

```

CREATE TABLE analytics (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  event_type TEXT NOT NULL, -- 'calendar_update', 'email_draft',
    'attachment_upload'
  success BOOLEAN NOT NULL,
  error_message TEXT,
  metadata TEXT, -- JSON con dettagli
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_analytics_type_date ON analytics(event_type,
  created_at);

-- Query utili
-- Eventi ultimi 7 giorni
SELECT event_type, COUNT(*) as total,
  SUM(CASE WHEN success THEN 1 ELSE 0 END) as successes
FROM analytics
WHERE created_at > datetime('now', '-7 days')
GROUP BY event_type;

-- Errori recenti
SELECT * FROM analytics
WHERE success = 0
ORDER BY created_at DESC
LIMIT 10;

```

---

## ☐ CHECKLIST FINALE

Prima di considerare il progetto completo:

### Funzionalità

- ☐ Funzione 1: Google Calendar funzionante
- ☐ Funzione 2: Gmail bozze funzionante
- ☐ Funzione 3: Upload/download allegati funzionante
- ☐ Funzione 4: Template con allegati default funzionanti

### UI/UX

- ☐ Dashboard riepilogativa
- ☐ Pagina gestione template
- ☐ Pagina gestione allegati
- ☐ Feedback visivo (loading, successo, errori)
- ☐ Responsive design (mobile-friendly)

### Sicurezza

- ☐ Tutti i secrets configurati correttamente
- ☐ Webhook authentication implementata
- ☐ Input validation su tutti gli endpoint
- ☐ CORS configurato correttamente
- ☐ Rate limiting implementato

## Performance

- ☐ Upload file grandi (>10MB) con signed URL
- ☐ Caching allegati frequenti
- ☐ Ottimizzazione query database
- ☐ Compression risposte API

## Documentazione

- ☐ README.md completo
- ☐ API documentation aggiornata
- ☐ Commenti nel codice
- ☐ Guide setup Google Cloud
- ☐ Guide configurazione Zapier

## Testing

- ☐ Test locali tutte le funzioni
- ☐ Test production tutte le funzioni
- ☐ Test integrazione Zapier
- ☐ Test edge cases e errori

## Deployment

- ☐ Production deployment completato
- ☐ DNS configurato (se custom domain)
- ☐ Monitoring attivo
- ☐ Backup database configurato

---

## 📌 PROSSIMI PASSI SUGGERITI

Dopo aver completato le 4 funzioni principali, considera:

### 1. Notifiche

- Email notification quando bozza creata
- Slack/Telegram notification per eventi importanti

### 2. Analytics Dashboard

- Grafici utilizzo template
- Statistiche allegati più usati
- Timeline attività

### 3. **Automazioni Avanzate**

- Auto-invio email dopo X giorni
- Promemoria follow-up automatici
- Sync bidirezionale con AirTable

### 4. **Integrazioni**

- WhatsApp Business API
- Telegram Bot
- Calendly integration

### 5. **Mobile App**

- Progressive Web App (PWA)
- Notifiche push
- Offline support

---

## **FINE DOCUMENTO SPECIFICHE TECNICHE**

*Ultimo aggiornamento: 2026-01-06 Versione: 2.2.6*