**Sistema di Stampa e Condivisione Documenti Clinici**

**1. MODULO GESTIONE STAMPE**

**1.1 Architettura Sistema di Stampa**

class PrintManagementSystem {

constructor() {

this.printQueue = new PrintQueue();

this.templateEngine = new TemplateEngine();

this.pdfGenerator = new PDFGenerator();

}

// Opzioni di stampa disponibili

printOptions = {

formats: {

full\_record: {

name: 'Cartella Clinica Completa',

sections: ['anagrafica', 'anamnesi', 'valutazioni', 'terapie', 'referti', 'consensi'],

defaultOrientation: 'portrait',

includeAttachments: true

},

summary: {

name: 'Riepilogo Paziente',

sections: ['anagrafica', 'diagnosi', 'piano\_terapeutico', 'progressi'],

defaultOrientation: 'portrait',

maxPages: 5

},

single\_report: {

name: 'Referto Singolo',

sections: ['selected\_report'],

defaultOrientation: 'portrait',

includeSignatures: true

},

therapy\_plan: {

name: 'Piano Terapeutico',

sections: ['diagnosi', 'obiettivi', 'sedute\_programmate', 'esercizi'],

defaultOrientation: 'landscape',

includeImages: true

},

discharge\_letter: {

name: 'Lettera di Dimissione',

sections: ['anagrafica', 'diagnosi', 'trattamento\_effettuato', 'risultati', 'raccomandazioni'],

defaultOrientation: 'portrait',

letterhead: true

},

invoice: {

name: 'Fattura/Ricevuta',

sections: ['dati\_fiscali', 'prestazioni', 'totali'],

defaultOrientation: 'portrait',

fiscal: true

}

},

customization: {

watermark: true,

headers: true,

footers: true,

pageNumbers: true,

logo: true,

digitalSignature: true,

qrCode: true

}

};

async preparePrintJob(patientId, options) {

const printJob = {

id: generateId(),

patientId: patientId,

timestamp: new Date(),

options: options,

status: 'preparing'

};

// Recupera dati necessari

const data = await this.collectData(patientId, options.sections);

// Applica template

const formatted = await this.templateEngine.format(data, options.template);

// Genera PDF

const pdf = await this.pdfGenerator.generate(formatted, options);

// Aggiungi watermark e sicurezza

if (options.watermark) {

pdf.addWatermark(options.watermarkText || 'COPIA CONFORME');

}

if (options.password) {

pdf.encrypt(options.password);

}

printJob.document = pdf;

printJob.status = 'ready';

return printJob;

}

}

**1.2 Generazione PDF Avanzata**

class AdvancedPDFGenerator:

def \_\_init\_\_(self):

self.engine = PDFEngine()

self.styles = self.load\_styles()

def generate\_clinical\_record(self, patient\_data, sections):

"""

Genera PDF cartella clinica con formattazione professionale

"""

pdf = PDF()

# Copertina

pdf.add\_page()

self.add\_cover\_page(pdf, patient\_data)

# Indice

pdf.add\_page()

toc = self.generate\_table\_of\_contents(sections)

pdf.add\_toc(toc)

# Sezioni

for section in sections:

pdf.add\_page()

if section == 'anagrafica':

self.add\_patient\_info(pdf, patient\_data)

elif section == 'anamnesi':

self.add\_anamnesis(pdf, patient\_data)

elif section == 'parametri\_vitali':

self.add\_vital\_signs\_chart(pdf, patient\_data)

elif section == 'terapie':

self.add\_therapy\_timeline(pdf, patient\_data)

elif section == 'referti':

self.add\_reports\_section(pdf, patient\_data)

elif section == 'immagini':

self.add\_images\_gallery(pdf, patient\_data)

elif section == 'consensi':

self.add\_consents(pdf, patient\_data)

# Footer con numerazione

self.add\_footers(pdf)

# QR Code per verifica

self.add\_qr\_verification(pdf)

return pdf

def add\_vital\_signs\_chart(self, pdf, data):

"""

Aggiunge grafici parametri vitali

"""

import matplotlib.pyplot as plt

# Crea grafico temperatura

fig, axes = plt.subplots(2, 2, figsize=(10, 8))

# Temperatura

axes[0, 0].plot(data['dates'], data['temperature'])

axes[0, 0].set\_title('Temperatura Corporea')

axes[0, 0].axhline(y=37, color='r', linestyle='--', alpha=0.5)

# Pressione

axes[0, 1].plot(data['dates'], data['bp\_systolic'], label='Sistolica')

axes[0, 1].plot(data['dates'], data['bp\_diastolic'], label='Diastolica')

axes[0, 1].set\_title('Pressione Arteriosa')

axes[0, 1].legend()

# Saturazione

axes[1, 0].plot(data['dates'], data['spo2'])

axes[1, 0].set\_title('Saturazione O₂')

axes[1, 0].axhline(y=95, color='r', linestyle='--', alpha=0.5)

# Dolore VAS

axes[1, 1].bar(data['dates'], data['pain\_vas'])

axes[1, 1].set\_title('Scala VAS Dolore')

# Salva come immagine temporanea

temp\_path = '/tmp/vitals\_chart.png'

plt.savefig(temp\_path, dpi=150, bbox\_inches='tight')

plt.close()

# Aggiungi al PDF

pdf.add\_image(temp\_path, x=10, y=30, w=190)

def add\_therapy\_timeline(self, pdf, data):

"""

Crea timeline visuale delle terapie

"""

from reportlab.graphics import renderPDF

from reportlab.graphics.shapes import Drawing

from reportlab.graphics.charts.lineplots import LinePlot

timeline = Drawing(400, 200)

# Crea timeline per ogni terapia

for therapy in data['therapies']:

# Aggiungi barra colorata per durata terapia

timeline.add(self.create\_therapy\_bar(

start=therapy['start\_date'],

end=therapy['end\_date'],

name=therapy['name'],

color=therapy['color']

))

renderPDF.draw(timeline, pdf.canvas, 50, 400)

**1.3 Sistema di Template Personalizzabili**

class TemplateSystem {

constructor() {

this.templates = new Map();

this.loadDefaultTemplates();

}

templates = {

'referto\_fisioterapico': {

layout: 'professional',

sections: [

{ type: 'header', content: '{clinic\_logo} {clinic\_info}' },

{ type: 'title', content: 'REFERTO DI VALUTAZIONE FISIOTERAPICA' },

{ type: 'patient\_info', fields: ['nome', 'cognome', 'cf', 'data\_nascita'] },

{ type: 'content', subsections: [

'anamnesi\_funzionale',

'esame\_obiettivo',

'test\_valutativi',

'diagnosi\_fisioterapica',

'piano\_trattamento',

'prognosi'

]},

{ type: 'signatures', required: ['fisioterapista', 'paziente'] },

{ type: 'footer', content: '{print\_date} - Pag {page}/{total\_pages}' }

],

styles: {

font: 'Arial',

fontSize: 11,

lineHeight: 1.5,

margins: { top: 20, bottom: 20, left: 15, right: 15 }

}

},

'cartella\_completa': {

layout: 'medical\_record',

coverPage: true,

tableOfContents: true,

sections: [

{ type: 'cover', design: 'professional\_medical' },

{ type: 'toc', maxDepth: 3 },

{ type: 'anagrafica', protected: true },

{ type: 'anamnesi', subsections: ['remota', 'prossima', 'familiare'] },

{ type: 'valutazioni', chronological: true },

{ type: 'terapie', includeDetails: true },

{ type: 'progressi', charts: true },

{ type: 'referti', sortBy: 'date\_desc' },

{ type: 'allegati', maxSize: '10MB' }

]

},

'summary\_discharge': {

layout: 'letter',

letterhead: true,

sections: [

{ type: 'letterhead', official: true },

{ type: 'recipient', fields: ['medico\_curante'] },

{ type: 'subject', content: 'Relazione di Dimissione' },

{ type: 'body', paragraphs: [

'diagnosi\_ingresso',

'trattamento\_effettuato',

'risultati\_ottenuti',

'situazione\_dimissione',

'raccomandazioni',

'follow\_up'

]},

{ type: 'closing', formal: true },

{ type: 'signatures', required: ['medico\_responsabile'] }

]

}

};

createCustomTemplate(name, config) {

// Editor visuale per template personalizzati

const customTemplate = {

name: name,

created: new Date(),

author: getCurrentUser(),

...config

};

this.templates.set(name, customTemplate);

this.saveTemplate(customTemplate);

return customTemplate;

}

}

**2. SISTEMA DI CONDIVISIONE MULTI-CANALE**

**2.1 Email Integration**

class EmailSharingSystem:

def \_\_init\_\_(self):

self.smtp\_config = SMTPConfig()

self.encryption = EmailEncryption()

async def send\_clinical\_document(self, recipient, document, options):

"""

Invia documento clinico via email con sicurezza

"""

# Prepara email

email = {

'to': recipient['email'],

'subject': self.generate\_subject(document, options),

'body': self.generate\_email\_body(document, options)

}

# Gestione allegati

if options.get('attach\_pdf'):

# Genera PDF protetto

pdf = await self.generate\_secure\_pdf(document)

# Cripta se richiesto

if options.get('encrypt'):

pdf = self.encryption.encrypt\_pdf(

pdf,

password=options.get('password') or self.generate\_otp()

)

# Invia password separatamente

if options.get('separate\_password'):

await self.send\_password\_sms(

recipient['phone'],

password

)

email['attachments'] = [{

'filename': f"{document['type']}\_{document['date']}.pdf",

'content': pdf,

'content\_type': 'application/pdf'

}]

# Link sicuro invece di allegato

elif options.get('secure\_link'):

link = await self.create\_secure\_link(document)

email['body'] += f"\n\nDocumento disponibile al link sicuro:\n{link}\n"

email['body'] += f"Il link scadrà tra {options.get('link\_expiry', 7)} giorni."

# Firma digitale email

if options.get('digital\_signature'):

email = self.sign\_email(email)

# Tracciamento apertura

if options.get('track\_opening'):

email['tracking\_pixel'] = self.add\_tracking\_pixel()

# Invio

result = await self.smtp\_config.send(email)

# Log per audit

self.log\_document\_sharing({

'type': 'email',

'recipient': recipient['email'],

'document': document['id'],

'timestamp': datetime.now(),

'status': result['status']

})

return result

def generate\_email\_body(self, document, options):

"""

Genera corpo email in base al tipo documento

"""

templates = {

'referto': """

Gentile {recipient\_name},

Le inviamo in allegato il referto della valutazione effettuata

in data {evaluation\_date}.

Il documento è protetto da password che le è stata inviata via SMS.

Per qualsiasi chiarimento, non esiti a contattarci.

Cordiali saluti,

{clinic\_name}

""",

'piano\_terapeutico': """

Gentile {recipient\_name},

Come concordato, le inviamo il piano terapeutico personalizzato.

Troverà in allegato:

- Programma delle sedute

- Esercizi da eseguire a casa

- Raccomandazioni

La prossima seduta è prevista per: {next\_appointment}

Cordiali saluti,

{therapist\_name}

""",

'dimissione': """

Gentile Dottore,

Le trasmettiamo la relazione di dimissione del paziente

{patient\_name}, che ha completato il ciclo riabilitativo

presso il nostro centro.

Restiamo a disposizione per eventuali chiarimenti.

Distinti saluti,

{medical\_director}

"""

}

return templates.get(document['type'], 'Documento clinico in allegato')

**2.2 WhatsApp Business Integration**

class WhatsAppIntegration {

constructor() {

this.whatsappAPI = new WhatsAppBusinessAPI();

this.mediaStorage = new MediaStorage();

}

async sendDocument(recipient, document, options) {

// Verifica consenso WhatsApp

if (!await this.checkWhatsAppConsent(recipient)) {

throw new Error('Consenso WhatsApp non presente');

}

// Prepara documento

let messageData = {

to: recipient.phone,

type: 'document'

};

// Gestione dimensioni file

const fileSize = document.size;

const maxSize = 100 \* 1024 \* 1024; // 100MB limite WhatsApp

if (fileSize > maxSize) {

// Comprimi o dividi documento

document = await this.compressDocument(document);

if (document.size > maxSize) {

// Invia link invece del file

return await this.sendDocumentLink(recipient, document);

}

}

// Upload documento

const mediaId = await this.mediaStorage.upload(document);

// Prepara messaggio WhatsApp

messageData.document = {

id: mediaId,

filename: `${document.type}\_${document.date}.pdf`,

caption: this.generateCaption(document, options)

};

// Template messaggio se disponibile

if (options.useTemplate) {

messageData = {

to: recipient.phone,

type: 'template',

template: {

name: options.templateName || 'document\_sharing',

language: { code: 'it' },

components: [

{

type: 'header',

parameters: [{

type: 'document',

document: { id: mediaId, filename: document.filename }

}]

},

{

type: 'body',

parameters: [

{ type: 'text', text: recipient.name },

{ type: 'text', text: document.type },

{ type: 'text', text: document.date }

]

}

]

}

};

}

// Invia messaggio

const result = await this.whatsappAPI.messages.send(messageData);

// Gestione password per documenti protetti

if (options.passwordProtected) {

await this.sendPasswordMessage(recipient, options.password);

}

// Conferma lettura

if (options.requestReadConfirmation) {

await this.scheduleReadConfirmationCheck(result.messageId);

}

// Log invio

this.logWhatsAppShare({

recipient: recipient.phone,

document: document.id,

messageId: result.messageId,

timestamp: new Date(),

status: result.status

});

return result;

}

async sendDocumentLink(recipient, document) {

// Genera link sicuro temporaneo

const secureLink = await this.generateSecureLink(document, {

expiry: '48h',

maxDownloads: 3,

requireAuth: true

});

// Messaggio con link

const message = {

to: recipient.phone,

type: 'text',

text: {

body: `📄 ${document.type}\n\n` +

`Il suo documento è disponibile al seguente link sicuro:\n` +

`${secureLink.url}\n\n` +

`⏰ Il link scadrà tra 48 ore\n` +

`🔒 Password: verrà inviata in un messaggio separato\n\n` +

`Per assistenza: ${this.clinic.phone}`

}

};

const result = await this.whatsappAPI.messages.send(message);

// Invia password separatamente

if (secureLink.password) {

setTimeout(() => {

this.sendPasswordMessage(recipient, secureLink.password);

}, 5000);

}

return result;

}

generateCaption(document, options) {

const captions = {

'referto': '📋 Referto valutazione fisioterapica',

'piano\_terapeutico': '📅 Piano terapeutico personalizzato',

'esercizi': '💪 Programma esercizi domiciliari',

'dimissione': '✅ Lettera di dimissione',

'consenso': '📝 Modulo consenso informato',

'fattura': '🧾 Fattura/Ricevuta'

};

let caption = captions[document.type] || '📄 Documento clinico';

if (options.includeInstructions) {

caption += '\n\n' + this.getInstructions(document.type);

}

return caption;

}

}

**2.3 Sistema di Link Sicuri**

class SecureLinkSystem:

def \_\_init\_\_(self):

self.storage = SecureStorage()

self.auth = AuthenticationService()

def create\_secure\_link(self, document, options=None):

"""

Crea link sicuro con scadenza e controlli accesso

"""

options = options or {}

# Genera token univoco

token = self.generate\_secure\_token()

# Salva documento in storage sicuro

storage\_path = self.storage.save\_temporary(

document,

token,

expiry=options.get('expiry', '7d')

)

# Crea record link

link\_record = {

'token': token,

'document\_id': document['id'],

'created': datetime.now(),

'expiry': self.calculate\_expiry(options.get('expiry', '7d')),

'max\_downloads': options.get('max\_downloads', 5),

'downloads\_count': 0,

'require\_auth': options.get('require\_auth', False),

'password': options.get('password'),

'allowed\_ips': options.get('allowed\_ips', []),

'watermark': options.get('watermark', True),

'log\_access': True

}

# Salva in database

self.db.save\_link(link\_record)

# Genera URL

base\_url = config.SECURE\_DOCUMENT\_URL

link\_url = f"{base\_url}/secure/{token}"

# Genera QR code

qr\_code = self.generate\_qr\_code(link\_url)

return {

'url': link\_url,

'token': token,

'expiry': link\_record['expiry'],

'password': link\_record['password'],

'qr\_code': qr\_code

}

async def access\_secure\_document(self, token, request\_info):

"""

Gestisce accesso a documento tramite link sicuro

"""

# Verifica validità link

link = self.db.get\_link(token)

if not link:

raise InvalidLinkError('Link non valido')

if link['expiry'] < datetime.now():

raise ExpiredLinkError('Link scaduto')

if link['downloads\_count'] >= link['max\_downloads']:

raise MaxDownloadsError('Numero massimo download raggiunto')

# Verifica autenticazione se richiesta

if link['require\_auth']:

auth\_result = await self.auth.verify\_access(request\_info)

if not auth\_result.valid:

raise AuthenticationError('Autenticazione richiesta')

# Verifica password se presente

if link['password']:

provided\_password = request\_info.get('password')

if not self.verify\_password(provided\_password, link['password']):

raise InvalidPasswordError('Password non corretta')

# Verifica IP se configurato

if link['allowed\_ips']:

if request\_info['ip'] not in link['allowed\_ips']:

raise IPNotAllowedError('Accesso non permesso da questo IP')

# Recupera documento

document = self.storage.get(link['document\_id'])

# Applica watermark se richiesto

if link['watermark']:

document = self.apply\_watermark(

document,

text=f"Scaricato da {request\_info['ip']} il {datetime.now()}"

)

# Log accesso

self.log\_access({

'token': token,

'ip': request\_info['ip'],

'user\_agent': request\_info.get('user\_agent'),

'timestamp': datetime.now(),

'success': True

})

# Incrementa contatore download

self.db.increment\_download\_count(token)

return document

**3. SISTEMA DI STAMPA BATCH E PROGRAMMATA**

**3.1 Print Queue Management**

class PrintQueueManager {

constructor() {

this.queue = [];

this.printers = this.detectPrinters();

this.processing = false;

}

async addBatchPrintJob(documents, options) {

const batchJob = {

id: generateBatchId(),

documents: documents,

options: options,

status: 'queued',

created: new Date(),

priority: options.priority || 'normal'

};

// Organizza documenti per tipo

const organized = this.organizeDocuments(documents);

// Crea job per ogni gruppo

for (const [type, docs] of Object.entries(organized)) {

const job = {

batchId: batchJob.id,

type: type,

documents: docs,

printer: this.selectPrinter(type, options),

settings: this.getPrintSettings(type, options),

status: 'pending'

};

this.queue.push(job);

}

// Processa coda se non in esecuzione

if (!this.processing) {

this.processQueue();

}

return batchJob;

}

async processQueue() {

this.processing = true;

while (this.queue.length > 0) {

// Ordina per priorità

this.queue.sort((a, b) => {

const priorities = { urgent: 3, high: 2, normal: 1, low: 0 };

return priorities[b.priority] - priorities[a.priority];

});

const job = this.queue.shift();

try {

// Prepara documento

const prepared = await this.prepareDocument(job);

// Invia a stampante

await this.sendToPrinter(prepared, job.printer, job.settings);

// Aggiorna stato

job.status = 'completed';

// Notifica completamento

this.notifyCompletion(job);

} catch (error) {

job.status = 'failed';

job.error = error.message;

// Retry logic

if (job.retryCount < 3) {

job.retryCount = (job.retryCount || 0) + 1;

job.status = 'retrying';

this.queue.push(job);

} else {

this.notifyFailure(job);

}

}

// Delay tra job

await this.delay(1000);

}

this.processing = false;

}

scheduleRecurringPrint(config) {

/\*\*

\* Pianifica stampe ricorrenti

\* Es: Report settimanali, riepilogo mensile

\*/

const schedule = {

id: generateScheduleId(),

name: config.name,

frequency: config.frequency, // daily, weekly, monthly

time: config.time, // HH:MM

dayOfWeek: config.dayOfWeek, // per weekly

dayOfMonth: config.dayOfMonth, // per monthly

query: config.query, // Query per selezionare documenti

template: config.template,

recipients: config.recipients,

active: true

};

// Registra in cron

this.cronManager.register(schedule, async () => {

const documents = await this.executeQuery(schedule.query);

if (documents.length > 0) {

const printJob = await this.addBatchPrintJob(documents, {

template: schedule.template,

priority: 'normal',

scheduled: true

});

// Invia anche via email se configurato

if (schedule.recipients?.email) {

await this.emailDocuments(documents, schedule.recipients.email);

}

}

});

return schedule;

}

}

**3.2 Print Preview e Editing**

class PrintPreviewSystem:

def \_\_init\_\_(self):

self.renderer = DocumentRenderer()

self.editor = InlineEditor()

def generate\_preview(self, document, options):

"""

Genera anteprima interattiva prima della stampa

"""

preview = {

'pages': [],

'metadata': {

'total\_pages': 0,

'paper\_size': options.get('paper\_size', 'A4'),

'orientation': options.get('orientation', 'portrait'),

'margins': options.get('margins', self.default\_margins)

}

}

# Renderizza ogni pagina

rendered = self.renderer.render(document, options)

for page\_num, page\_content in enumerate(rendered.pages):

page\_preview = {

'number': page\_num + 1,

'content': page\_content,

'thumbnail': self.generate\_thumbnail(page\_content),

'editable\_regions': self.identify\_editable\_regions(page\_content)

}

preview['pages'].append(page\_preview)

preview['metadata']['total\_pages'] = len(preview['pages'])

# Calcola statistiche

preview['statistics'] = {

'ink\_usage': self.estimate\_ink\_usage(rendered),

'paper\_sheets': self.calculate\_paper\_sheets(

len(preview['pages']),

options.get('duplex', False)

),

'estimated\_time': self.estimate\_print\_time(preview)

}

return preview

def apply\_last\_minute\_edits(self, preview, edits):

"""

Applica modifiche dell'ultimo minuto prima della stampa

"""

for edit in edits:

if edit['type'] == 'text\_replace':

preview = self.replace\_text(

preview,

edit['old\_text'],

edit['new\_text']

)

elif edit['type'] == 'highlight':

preview = self.add\_highlight(

preview,

edit['page'],

edit['region'],

edit['color']

)

elif edit['type'] == 'annotation':

preview = self.add\_annotation(

preview,

edit['page'],

edit['position'],

edit['text']

)

elif edit['type'] == 'redact':

preview = self.redact\_content(

preview,

edit['page'],

edit['region']

)

return preview

**4. INTERFACCIA UTENTE STAMPA E CONDIVISIONE**

**4.1 Print Dialog Component**

const PrintDialog = ({ patient, documents, onPrint, onShare }) => {

const [selectedDocs, setSelectedDocs] = useState([]);

const [printOptions, setPrintOptions] = useState({

format: 'full\_record',

orientation: 'portrait',

paperSize: 'A4',

copies: 1,

duplex: false,

color: false,

watermark: true,

includeSignatures: true

});

const [shareOptions, setShareOptions] = useState({

method: 'email',

recipient: '',

encrypt: true,

passwordMethod: 'sms',

linkExpiry: '7days',

allowDownload: true,

trackAccess: true

});

const documentTypes = {

'Cartella Completa': ['all'],

'Solo Anagrafica': ['patient\_info'],

'Anamnesi': ['anamnesis'],

'Valutazioni': ['assessments'],

'Terapie': ['therapies'],

'Referti': ['reports'],

'Consensi': ['consents'],

'Immagini': ['images'],

'Personalizzato': ['custom']

};

return (

<Dialog open={true} maxWidth="md" fullWidth>

<DialogTitle>

<Tabs value={activeTab} onChange={setActiveTab}>

<Tab label="Stampa" icon={<Print />} />

<Tab label="Email" icon={<Mail />} />

<Tab label="WhatsApp" icon={<MessageSquare />} />

<Tab label="Link Sicuro" icon={<Link />} />

</Tabs>

</DialogTitle>

<DialogContent>

{activeTab === 0 && (

<PrintOptionsPanel>

<SectionSelector

types={documentTypes}

selected={selectedDocs}

onChange={setSelectedDocs}

/>

<PrintSettings>

<FormControl>

<InputLabel>Formato</InputLabel>

<Select value={printOptions.format}>

<MenuItem value="full\_record">Cartella Completa</MenuItem>

<MenuItem value="summary">Riepilogo</MenuItem>

<MenuItem value="medical\_report">Referto Medico</MenuItem>

<MenuItem value="discharge">Lettera Dimissione</MenuItem>

</Select>

</FormControl>

<FormControl>

<InputLabel>Orientamento</InputLabel>

<ToggleButtonGroup value={printOptions.orientation}>

<ToggleButton value="portrait">Verticale</ToggleButton>

<ToggleButton value="landscape">Orizzontale</ToggleButton>

</ToggleButtonGroup>

</FormControl>

<FormControlLabel

control={<Checkbox checked={printOptions.watermark} />}

label="Aggiungi Watermark"

/>

<FormControlLabel

control={<Checkbox checked={printOptions.includeSignatures} />}

label="Includi Firme Digitali"

/>

</PrintSettings>

<PreviewArea>

<PrintPreview

documents={selectedDocs}

options={printOptions}

/>

</PreviewArea>

</PrintOptionsPanel>

)}

{activeTab === 1 && (

<EmailOptionsPanel>

<TextField

label="Destinatario Email"

value={shareOptions.recipient}

onChange={(e) => setShareOptions({

...shareOptions,

recipient: e.target.value

})}

fullWidth

/>

<FormControlLabel

control={<Switch checked={shareOptions.encrypt} />}

label="Cripta documento (PDF protetto)"

/>

{shareOptions.encrypt && (

<RadioGroup value={shareOptions.passwordMethod}>

<FormControlLabel

value="sms"

control={<Radio />}

label="Invia password via SMS"

/>

<FormControlLabel

value="email"

control={<Radio />}

label="Password nella stessa email"

/>

<FormControlLabel

value="custom"

control={<Radio />}

label="Password personalizzata"

/>

</RadioGroup>

)}

<TextField

label="Messaggio personalizzato"

multiline

rows={4}

fullWidth

/>

</EmailOptionsPanel>

)}

{activeTab === 2 && (

<WhatsAppOptionsPanel>

<PhoneInput

label="Numero WhatsApp"

value={shareOptions.phone}

onChange={(value) => setShareOptions({

...shareOptions,

phone: value

})}

/>

<Alert severity="info">

Il documento verrà inviato come PDF tramite WhatsApp Business.

Limite dimensione: 100MB

</Alert>

<FormControlLabel

control={<Checkbox />}

label="Richiedi conferma lettura"

/>

<FormControlLabel

control={<Checkbox />}

label="Invia anche link di backup"

/>

</WhatsAppOptionsPanel>

)}

{activeTab === 3 && (

<SecureLinkPanel>

<Select

label="Validità link"

value={shareOptions.linkExpiry}

>

<MenuItem value="24h">24 ore</MenuItem>

<MenuItem value="48h">48 ore</MenuItem>

<MenuItem value="7days">7 giorni</MenuItem>

<MenuItem value="30days">30 giorni</MenuItem>

</Select>

<TextField

label="Max download"

type="number"

value={shareOptions.maxDownloads}

InputProps={{ inputProps: { min: 1, max: 100 } }}

/>

<FormControlLabel

control={<Switch checked={shareOptions.requireAuth} />}

label="Richiedi autenticazione"

/>

<FormControlLabel

control={<Switch checked={shareOptions.trackAccess} />}

label="Traccia accessi"

/>

<GeneratedLink>

{generatedLink && (

<div>

<Typography variant="caption">Link generato:</Typography>

<CopyableLink value={generatedLink} />

<QRCode value={generatedLink} size={150} />

</div>

)}

</GeneratedLink>

</SecureLinkPanel>

)}

</DialogContent>

<DialogActions>

<Button onClick={onCancel}>Annulla</Button>

<Button

onClick={() => handleAction()}

variant="contained"

startIcon={getActionIcon()}

>

{getActionLabel()}

</Button>

</DialogActions>

</Dialog>

);

};

**5. AUDIT E TRACCIABILITÀ**

**5.1 Sistema di Log per Stampe e Condivisioni**

-- Tabella log stampe

CREATE TABLE print\_log (

id SERIAL PRIMARY KEY,

document\_id INTEGER,

patient\_id INTEGER,

user\_id INTEGER,

print\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

document\_type VARCHAR(50),

pages\_printed INTEGER,

printer\_name VARCHAR(100),

print\_options JSON,

cost\_estimate DECIMAL(10,2),

status ENUM('queued', 'printing', 'completed', 'failed'),

error\_message TEXT,

ip\_address VARCHAR(45),

workstation VARCHAR(100)

);

-- Tabella log condivisioni

CREATE TABLE sharing\_log (

id SERIAL PRIMARY KEY,

document\_id INTEGER,

patient\_id INTEGER,

shared\_by INTEGER,

shared\_with VARCHAR(255),

sharing\_method ENUM('email', 'whatsapp', 'link', 'fax'),

shared\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

document\_encrypted BOOLEAN,

password\_sent\_via VARCHAR(50),

link\_token VARCHAR(255),

link\_expiry TIMESTAMP,

max\_downloads INTEGER,

current\_downloads INTEGER DEFAULT 0,

last\_access TIMESTAMP,

access\_log JSON,

revoked BOOLEAN DEFAULT FALSE,

revoked\_date TIMESTAMP,

revoked\_by INTEGER,

revoke\_reason TEXT

);

-- Vista statistiche utilizzo

CREATE VIEW sharing\_statistics AS

SELECT

DATE(shared\_date) as date,

sharing\_method,

COUNT(\*) as total\_shares,

COUNT(DISTINCT patient\_id) as unique\_patients,

COUNT(DISTINCT shared\_by) as unique\_users,

AVG(current\_downloads) as avg\_downloads,

SUM(CASE WHEN revoked THEN 1 ELSE 0 END) as revoked\_count

FROM sharing\_log

GROUP BY DATE(shared\_date), sharing\_method;

**5.2 Dashboard Monitoraggio Stampe e Condivisioni**

const PrintShareDashboard = {

stats: {

today: {

prints: 45,

emails: 23,

whatsapp: 18,

secureLinks: 12

},

month: {

totalDocuments: 1243,

uniquePatients: 456,

averagePerDay: 41,

peakDay: { date: '2024-01-15', count: 89 }

},

costs: {

paperUsed: '2 risme',

inkUsage: '15%',

estimatedCost: '€ 45.30'

},

security: {

encryptedShares: '94%',

linksExpired: 145,

unauthorizedAttempts: 3,

successfulAccesses: '98%'

}

},

alerts: [

{

type: 'warning',

message: 'Stampante Piano 2 - Toner in esaurimento',

action: 'Ordinare ricambio'

},

{

type: 'info',

message: '15 link sicuri in scadenza oggi',

action: 'Notifica automatica inviata'

}

],

recentActivity: [

{

time: '10:45',

user: 'Dr. Bianchi',

action: 'Stampato referto',

patient: 'Mario Rossi',

details: '5 pagine, A4'

},

{

time: '10:30',

user: 'Ft. Verdi',

action: 'Inviato via WhatsApp',

patient: 'Anna Blu',

details: 'Piano esercizi, PDF 2.3MB'

}

]

};

**6. BEST PRACTICES E SICUREZZA**

**6.1 Linee Guida Sicurezza**

security\_guidelines = {

'encryption': {

'always\_encrypt': ['referti', 'cartelle\_complete', 'dati\_sensibili'],

'password\_complexity': 'min 8 char, 1 maiuscola, 1 numero, 1 speciale',

'password\_delivery': 'sempre canale separato (SMS/chiamata)',

'encryption\_algorithm': 'AES-256'

},

'access\_control': {

'link\_expiry': 'max 7 giorni per documenti clinici',

'max\_downloads': 'max 5 per documenti sensibili',

'ip\_restriction': 'opzionale per extra sicurezza',

'authentication': 'richiesta per cartelle complete'

},

'audit': {

'log\_everything': True,

'retention\_period': '10 anni',

'tamper\_proof': 'hash chain per integrità log',

'regular\_reviews': 'mensile'

},

'gdpr\_compliance': {

'consent\_required': True,

'right\_to\_deletion': True,

'data\_portability': True,

'breach\_notification': '72 ore'

}

}

**6.2 Procedure di Emergenza**

const emergencyProcedures = {

dataBreachDetected: async (incident) => {

// 1. Isola il problema

await revokeAllActiveLinks(incident.affectedDocuments);

// 2. Notifica immediata

await notifySecurityTeam(incident);

await notifyAffectedPatients(incident);

// 3. Documenta

await createIncidentReport(incident);

// 4. Remediation

await implementSecurityPatch(incident.vulnerability);

// 5. Follow-up

await scheduleSecurityAudit();

},

unauthorizedAccessAttempt: async (attempt) => {

// Log dettagliato

await logSecurityEvent(attempt);

// Blocco temporaneo

if (attempt.count > 3) {

await blockIP(attempt.ip, '24h');

}

// Alert amministratore

await alertAdmin(attempt);

}

};