sebastien clem initial import: facturier demo app 9b27b35 · 3 weeks ago

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Copyright (C) 2026 Sébastien
# This program is licensed under the GNU GPL v3.

import json
import re
import sys
from collections import defaultdict
from decimal import Decimal, ROUND_HALF_UP
from pathlib import Path

from reportlab.lib.pagesizes import A4
from reportlab.lib.styles import ParagraphStyle, getSampleStyleSheet
from reportlab.lib.units import mm
from reportlab.pdfgen import canvas
from reportlab.platypus import Paragraph

Q2 = Decimal("0.01")
PAGE_WIDTH, PAGE_HEIGHT = A4


def q2(value) -> Decimal:
    return Decimal(str(value)).quantize(Q2, rounding=ROUND_HALF_UP)


def format_money(value: Decimal, currency: str = "EUR") -> str:
    s = f"{q2(value):,.2f}".replace(",", "X").replace(".", ",").replace("X", " ")
    return f"{s} €" if currency == "EUR" else f"{s} {currency}"


def format_quantity(value) -> str:
    d = Decimal(str(value))
    if d == d.to_integral():
        return str(int(d))
    return str(d).replace(".", ",")


def normalize_country_code(value: str, default="FR") -> str:
    value = (value or default).strip().upper()
    return value if len(value) == 2 and value.isalpha() else default


def split_postcode_city(value: str):
    value = (value or "").strip()
    if not value:
        return "", ""
```

facturier-app / scripts / generate_invoice_pdf.py ↑ Top

Code | Blame 483 lines (368 loc) · 16.7 KB Raw

def draw_invoice_info(c: canvas.Canvas, data: dict, x: float, y: float, w: float):

```python

def normalize_party_address(party: dict) -> dict:
    street = (party.get("street") or "").strip()
    postcode = (party.get("postcode") or "").strip()
    city = (party.get("city") or "").strip()
    country_code = normalize_country_code(party.get("country_code"), "FR")

    address_lines = party.get("address_lines") or []
    if isinstance(address_lines, str):
        address_lines = [line.strip() for line in address_lines.splitlines() if line.strip()]
    else:
        address_lines = [str(line).strip() for line in address_lines if str(line).strip()]

    if not street and address_lines:
        street = address_lines[0]
    if (not postcode or not city) and len(address_lines) >= 2:
        p, c = split_postcode_city(address_lines[1])
        postcode = postcode or p
        city = city or c
    if len(address_lines) >= 3 and not party.get("country_code"):
        country_code = normalize_country_code(address_lines[2], country_code)

    result = dict(party)
    result["street"] = street
    result["postcode"] = postcode
    result["city"] = city
    result["country_code"] = country_code
    result["address_lines"] = [part for part in [street, " ".join([v for v in [postcode, city] i
    return result


def compute_invoice(data: dict) -> dict:
    currency = data["document"].get("currency", "EUR")
    tax_map = defaultdict(lambda: {"base_ht": Decimal("0.00"), "tax_amount": Decimal("0.00")})
    computed_lines = []
    total_ht = Decimal("0.00")

    for idx, line in enumerate(data["lines"], start=1):
        qty = Decimal(str(line["quantity"]))
        unit_price = Decimal(str(line["unit_price_ht"]))
        vat_rate = Decimal(str(line["vat_rate"]))

        line_total = q2(qty * unit_price)
        line_tax = q2(line_total * vat_rate / Decimal("100"))

        computed_line = dict(line)
        computed_line["code"] = line.get("code") or str(idx).zfill(3)
        computed_line["line_total_ht"] = line_total
        computed_line["line_tax"] = line_tax
        computed_lines.append(computed_line)

        total_ht += line_total
        tax_map[vat_rate]["base_ht"] += line_total
        tax_map[vat_rate]["tax_amount"] += line_tax

    total_ht = q2(total_ht)
    total_tax = q2(sum(v["tax_amount"] for v in tax_map.values()))
    total_ttc = q2(total_ht + total_tax)

    vat_summary = []
    for rate in sorted(tax_map.keys()):
        vat_summary.append({
            "rate": q2(rate),
            "base_ht": q2(tax_map[rate]["base_ht"]),
            "tax_amount": q2(tax_map[rate]["tax_amount"]),
        })

    return {
        "currency": currency,
        "lines": computed_lines,
        "vat_summary": vat_summary,
        "totals": {
            "total_lines_ht": total_ht,
            "tax_base": total_ht,
            "tax_total": total_tax,
            "total_ttc": total_ttc,
            "due_payable": total_ttc,
        },
    }


def build_styles():
    styles = getSampleStyleSheet()
    return {
        "desc": ParagraphStyle("Desc", parent=styles["BodyText"], fontName="Times-Roman", fontSi
        "note": ParagraphStyle("Note", parent=styles["BodyText"], fontName="Times-Roman", fontSi
    }


def wrap_paragraph(text: str, width: float, style: ParagraphStyle):
    p = Paragraph((text or "").replace("\n", "<br/>"), style)
    _, h = p.wrap(width, 10_000)
    return p, h


def draw_box(c: canvas.Canvas, x, y_top, w, h):
    c.rect(x, y_top - h, w, h, stroke=1, fill=0)


def draw_section_title(c: canvas.Canvas, x, y, text: str, size: int = 16):
    c.setFont("Times-Bold", size)
    c.drawString(x, y, text)


def draw_label_value(c, x, y, label, value, label_width=85, font_size=12):
    c.setFont("Times-Bold", font_size)
    c.drawString(x, y, label)
    c.setFont("Times-Roman", font_size)
    c.drawString(x + label_width, y, value or "")


def ensure_space(c: canvas.Canvas, y: float, needed: float, margin_bottom: float, redraw_header=
    if y - needed >= margin_bottom:
        return y
    c.showPage()
    return redraw_header(c) if redraw_header else PAGE_HEIGHT - 20 * mm


def split_address_lines(value):
    if isinstance(value, list):
        return value
    if not value:
        return []
    return str(value).splitlines()


def estimate_party_height(party: dict) -> float:
    line_h = 15
    top_pad = 18
    bottom_pad = 10
    title_block = 18
    party = normalize_party_address(party)
    address_lines = split_address_lines(party.get("address_lines"))
    total_lines = 1 + 2 + len(address_lines) + 2
    content_h = total_lines * line_h
    return max(60 * mm, top_pad + title_block + content_h + bottom_pad)


def estimate_info_height(document: dict) -> float:
    rows = 4 if document.get("currency") else 3
    return max(26 * mm, 14 + rows * 15 + 8)


def estimate_notes_height(notes: list[str], style: ParagraphStyle, width: float) -> float:
    total = 18
    for note in notes:
        _, h = wrap_paragraph(note, width - 16, style)
        total += h + 4
    total += 8
    return max(22 * mm, total)


def estimate_payment_height(payment: dict) -> float:
    rows = 1 if payment.get("iban") else 0
    return max(14 * mm, 14 + rows * 15 + 8)


def draw_page_header(c: canvas.Canvas, data: dict, margin_x: float, margin_top: float):
    y = PAGE_HEIGHT - margin_top
    c.setFont("Times-Bold", 22)
    c.drawString(margin_x, y, data["seller"]["name"])
    y -= 18
    c.setFont("Times-Roman", 14)
    c.drawString(margin_x, y, data["document"].get("title", "Facture"))
    return y - 28


def draw_contacts(c: canvas.Canvas, data: dict, x: float, y: float, w: float):
    left_h = estimate_party_height(data["seller"])
    right_h = estimate_party_height(data["buyer"])
    h = max(left_h, right_h)

    draw_section_title(c, x, y, "Contacts")
    y -= 10
    draw_box(c, x, y, w, h)
    col_w = w / 2
    c.line(x + col_w, y, x + col_w, y - h)

    def draw_party(px, py, title, party):
        party = normalize_party_address(party)
        c.setFont("Times-Bold", 18)
        c.drawString(px, py - 20, title)

        cy = py - 38
        draw_label_value(c, px, cy, "Entreprise :", party.get("name", "")); cy -= 15
        draw_label_value(c, px, cy, "SIREN :", party.get("siren") or party.get("siret", "")); cy
        draw_label_value(c, px, cy, "TVA :", party.get("vat_number", "")); cy -= 15

        c.setFont("Times-Bold", 12)
        c.drawString(px, cy, "Adresse :")
        c.setFont("Times-Roman", 12)
        addr_x = px + 55
        for line in split_address_lines(party.get("address_lines")):
            c.drawString(addr_x, cy, line)
            cy -= 15

        draw_label_value(c, px, cy, "Email :", party.get("email", ""), label_width=55); cy -= 15
        draw_label_value(c, px, cy, "Téléphone :", party.get("phone", ""), label_width=55)

    draw_party(x + 8, y, "ÉMETTEUR", data["seller"])
    draw_party(x + col_w + 8, y, "CLIENT", data["buyer"])
    return y - h - 26


def draw_invoice_info(c: canvas.Canvas, data: dict, x: float, y: float, w: float):
    h = estimate_info_height(data["document"])
    draw_section_title(c, x, y, "Informations facture")
    y -= 8
    draw_box(c, x, y, w, h)

    cy = y - 18
    draw_label_value(c, x + 8, cy, "Numéro de facture :", data["document"]["invoice_number"], la
    draw_label_value(c, x + 8, cy, "Date de facture :", data["document"]["issue_date"], label_wi
    draw_label_value(c, x + 8, cy, "Date d'échéance :", data["document"]["due_date"], label_widt

    if data["document"].get("currency"):
        cy -= 15
        draw_label_value(c, x + 8, cy, "Devise :", data["document"]["currency"], label_width=120

    return y - h - 24


def build_line_table_columns(table_w: float):
    fixed = {"code": 24 * mm, "qty": 18 * mm, "unit": 14 * mm, "unit_price": 26 * mm, "vat": 18
    description_w = table_w - sum(fixed.values())
    return [fixed["code"], description_w, fixed["qty"], fixed["unit"], fixed["unit_price"], fixe


def draw_lines_table(c: canvas.Canvas, calc: dict, x: float, y: float, w: float, margin_bottom:
    styles = build_styles()
    desc_style = styles["desc"]

    draw_section_title(c, x, y, "Lignes de factures")
    y -= 8

    col_widths = build_line_table_columns(w)
    headers = ["Code", "Description", "Quantité", "Unité", "Prix unitaire HT", "TVA", "Total HT"

    x_positions = [x]
    for cw in col_widths:
        x_positions.append(x_positions[-1] + cw)

    header_h = 22
    table_top_y = y

    prepared = []
    for line in calc["lines"]:
        para, para_h = wrap_paragraph(line["description"], col_widths[1] - 8, desc_style)
        row_h = max(24, para_h + 8)
        prepared.append((line, para, para_h, row_h))

    def draw_table_header(top_y):
        c.rect(x, top_y - header_h, w, header_h, stroke=1, fill=0)
        for xp in x_positions[1:-1]:
            c.line(xp, top_y, xp, top_y - header_h)
        c.setFont("Times-Roman", 10.5)
        header_y = top_y - 15
        for i, header in enumerate(headers):
            c.drawString(x_positions[i] + 4, header_y, header)

    draw_table_header(table_top_y)
    current_y = table_top_y - header_h

    for line, para, para_h, row_h in prepared:
        if current_y - row_h < margin_bottom:
            c.showPage()
            y = redraw_header(c)
            draw_section_title(c, x, y, "Lignes de factures (suite)")
            y -= 8
            draw_table_header(y)
            current_y = y - header_h

        c.rect(x, current_y - row_h, w, row_h, stroke=1, fill=0)
        for xp in x_positions[1:-1]:
            c.line(xp, current_y, xp, current_y - row_h)

        c.setFont("Times-Roman", 10.5)
        c.drawString(x_positions[0] + 4, current_y - 15, line["code"])
        para.drawOn(c, x_positions[1] + 4, current_y - para_h - 5)
        c.drawRightString(x_positions[3] - 4, current_y - 15, format_quantity(line["quantity"]))
        c.drawString(x_positions[3] + 4, current_y - 15, line.get("unit", ""))
        c.drawRightString(x_positions[5] - 4, current_y - 15, format_money(line["unit_price_ht"]
        c.drawRightString(x_positions[6] - 4, current_y - 15, f"{q2(Decimal(str(line['vat_rate']
        c.drawRightString(x_positions[7] - 4, current_y - 15, format_money(line["line_total_ht"]

        current_y -= row_h

    return current_y - 24


def draw_vat_and_totals(c: canvas.Canvas, calc: dict, x: float, y: float, page_w: float, margin_
    vat_w = 90 * mm
    totals_w = 78 * mm
    gap = 8 * mm

    vat_col_widths = [30 * mm, 22 * mm, 43 * mm]
    vat_headers = ["Base HT", "Taux TVA", "Montant TVA"]
    vat_row_h = 18
    vat_h = vat_row_h * (1 + len(calc["vat_summary"]))

    totals_rows = [
        ("Total lignes HT", calc["totals"]["total_lines_ht"]),
        ("Base taxable", calc["totals"]["tax_base"]),
        ("TVA", calc["totals"]["tax_total"]),
        ("Total TTC", calc["totals"]["total_ttc"]),
        ("Net à payer", calc["totals"]["due_payable"]),
    ]
    totals_row_h = 21
    totals_h = totals_row_h * len(totals_rows)

    block_h = max(vat_h + 24, totals_h + 24)
    y = ensure_space(c, y, block_h + 10, margin_bottom, redraw_header)

    draw_section_title(c, x, y, "Détails TVA")
    vat_top = y - 8
    c.rect(x, vat_top - vat_h, vat_w, vat_h, stroke=1, fill=0)

    running = x
    for cw in vat_col_widths[:-1]:
        running += cw
        c.line(running, vat_top, running, vat_top - vat_h)
    for i in range(1, 1 + len(calc["vat_summary"])):
        c.line(x, vat_top - vat_row_h * i, x + vat_w, vat_top - vat_row_h * i)

    c.setFont("Times-Roman", 10.5)
    header_y = vat_top - 15
    running = x
    for i, h in enumerate(vat_headers):
        c.drawString(running + 4, header_y, h)
        running += vat_col_widths[i]

    for i, row in enumerate(calc["vat_summary"], start=1):
        row_y = vat_top - vat_row_h * i - 15
        c.drawRightString(x + vat_col_widths[0] - 4, row_y, format_money(row["base_ht"], calc["c
        c.drawRightString(x + vat_col_widths[0] + vat_col_widths[1] - 4, row_y, f"{row['rate']}
        c.drawRightString(x + vat_w - 4, row_y, format_money(row["tax_amount"], calc["currency"]

    totals_x = x + vat_w + gap
    draw_section_title(c, totals_x, y, "Totaux")
    totals_top = y - 8
    c.rect(totals_x, totals_top - totals_h, totals_w, totals_h, stroke=1, fill=0)

    label_w = 42 * mm
    c.line(totals_x + label_w, totals_top, totals_x + label_w, totals_top - totals_h)
    for i in range(1, len(totals_rows)):
        c.line(totals_x, totals_top - totals_row_h * i, totals_x + totals_w, totals_top - totals

    for i, (label, value) in enumerate(totals_rows):
        row_y = totals_top - totals_row_h * i - 15
        c.setFont("Times-Roman", 10.5)
        c.drawString(totals_x + 4, row_y, label)
        c.drawRightString(totals_x + totals_w - 4, row_y, format_money(value, calc["currency"]))

    return min(vat_top - vat_h, totals_top - totals_h) - 24


def draw_legal_notes(c: canvas.Canvas, data: dict, x: float, y: float, w: float, margin_bottom:
    styles = build_styles()
    note_style = styles["note"]
    notes = data.get("legal_notes", [])
    h = estimate_notes_height(notes, note_style, w)
    y = ensure_space(c, y, h + 12, margin_bottom, redraw_header)

    draw_section_title(c, x, y, "Mentions légales")
    y -= 8
    draw_box(c, x, y, w, h)

    cy = y - 16
    for note in notes:
        p, p_h = wrap_paragraph(note, w - 16, note_style)
        p.drawOn(c, x + 8, cy - p_h)
        cy -= p_h + 4

    return y - h - 20


def draw_payment(c: canvas.Canvas, data: dict, x: float, y: float, w: float, margin_bottom: floa
    h = estimate_payment_height(data.get("payment", {}))
    y = ensure_space(c, y, h + 12, margin_bottom, redraw_header)

    draw_section_title(c, x, y, "Paiement")
    y -= 8
    draw_box(c, x, y, w, h)
    draw_label_value(c, x + 8, y - 18, "IBAN :", data.get("payment", {}).get("iban", ""), label_
    return y - h - 16


def generate_pdf(data: dict, output_path: Path):
    data = dict(data)
    data["seller"] = normalize_party_address(data["seller"])
    data["buyer"] = normalize_party_address(data["buyer"])
    calc = compute_invoice(data)

    c = canvas.Canvas(str(output_path), pagesize=A4)

    margin_x = 18 * mm
    margin_top = 20 * mm
    margin_bottom = 16 * mm
    content_w = PAGE_WIDTH - 2 * margin_x

    def redraw_header(local_canvas):
        return draw_page_header(local_canvas, data, margin_x, margin_top)

    y = redraw_header(c)
    y = draw_contacts(c, data, margin_x, y, content_w)
    y = draw_invoice_info(c, data, margin_x, y, content_w)
    y = draw_lines_table(c, calc, margin_x, y, content_w, margin_bottom, redraw_header)
    y = draw_vat_and_totals(c, calc, margin_x, y, content_w, margin_bottom, redraw_header)
    y = draw_legal_notes(c, data, margin_x, y, content_w, margin_bottom, redraw_header)
    y = draw_payment(c, data, margin_x, y, content_w, margin_bottom, redraw_header)

    c.save()


def main():
    if len(sys.argv) != 3:
        print("Usage: python generate_invoice_pdf.py invoice.json output.pdf")
        sys.exit(1)

    json_path = Path(sys.argv[1])
    output_path = Path(sys.argv[2])

    with json_path.open("r", encoding="utf-8") as f:
        data = json.load(f)

    generate_pdf(data, output_path)
    print(f"PDF généré : {output_path}")


if __name__ == "__main__":
    main()
```