

Printservice Standalone Java 25 v2.3

Eigenständiger Druckservice ohne GlassFish/Payara. Die Anwendung läuft als normale Java-Applikation oder über WinSW als Windows-Dienst und stellt ein Webfrontend plus REST-API bereit.

Funktionen

- eigener HTTP-Server aus dem JDK
- Webfrontend unter `http://localhost:8080/`
- REST-API unter `http://localhost:8080/api`
- Drucker-Aliase für `WINDOWS_SPOOLER` und `RAW_TCP`
- Windows-/LocalSystem-Assistent für Dienstbetrieb
- HTML-Vorlagen mit Online-Editor
- HTML-Tabellen über `{{#each:items}} ... {{/each}}`
- HTML -> PDF -> Druck
- PDF als Vorlage mit Koordinatenfeldern (`PDF_TEMPLATE`)
- PDF-Tabellen aus JSON-Arrays
- ZPL/EPL/Text RAW-Druck
- DOCX-Vorlagen rendern und herunterladen
- DOCX -> PDF -> Druck optional über `DOCX4J_F0` ohne LibreOffice
- Barcode, QR-Code, DataMatrix und weitere ZXing-Formate
- optionale Authentifizierung: wenn deaktiviert, kann jeder per REST drucken
- Benutzer-/Tokenverwaltung im Webfrontend
- persistente Druckjob-Historie / Druckwarteschlange für Redruck
- Filterung der Druckhistorie nach User, Status, Drucker und Vorlage
- Admin-Vollansicht der Warteschlange mit `full=true`
- produktionsnahe Vorlagenstruktur unter `templates/<templateName>/...`

Zielpattform

- Java 25
- Maven 3.9 oder neuer empfohlen
- Windows, Linux oder macOS
- Windows-Dienstbetrieb über WinSW

Build

`mvn clean package`

Ergebnis:

`target/printservice-standalone.jar`

Start als Konsole

Windows:

```
java -Dprintservice.config.dir=C:\printservice -jar target\printservice-standalone.jar --host=localhost
```

Linux:

```
java -Dprintservice.config.dir=/opt/printservice -jar target/printservice-standalone.jar --host=localhost
```

Danach:

<http://localhost:8080/>

<http://localhost:8080/api/diagnostics/system>

Konfigurationsstruktur

Empfohlen unter Windows:

```
C:\printservice
  auth.json
  printers.json
  queue
    print-jobs.json
  templates
    <templateName>
      template.json
      document.*
      fields.json
      sample-data.json
      print-request.json
  spool
  archive
  preview
  logs
```

Ohne `-Dprintservice.config.dir` nutzt die Anwendung:

```
%USERPROFILE%\printservice
```

Windows-Service mit WinSW

```
mvn clean package
```

```
.\scripts\install-service.ps1 -InstallDir C:\printservice-app -ConfigDir C:\printservice -Port 8080
```

Danach WinSW-x64.exe als `C:\printservice-app\printservice.exe` ablegen

und starten:

```
cd C:\printservice-app
.\printservice.exe install
```

```
.\printservice.exe start
.\printservice.exe status
```

Authentifizierung

Standardmäßig ist Auth deaktiviert. Dann kann jeder, der `/api/print` erreicht, drucken. Das ist für Tests praktisch, aber produktiv nur in geschlossenen Netzen sinnvoll.

```
{
  "enabled": false,
  "protectPrint": true,
  "protectRenderedDownload": true,
  "protectPreview": false,
  "protectAdmin": false,
  "headerName": "X-Printservice-Token",
  "tokens": []
}
```

Empfohlener Ablauf im Webfrontend

1. Webfrontend öffnen.
2. Bereich **Authentifizierung, Benutzer und Token**.
3. Erst einen Admin-User anlegen, Scope ALL.
4. Token kopieren und sicher speichern.
5. Danach Auth aktivieren.
6. Optional `protectAdmin` aktivieren.
7. Für Maschinen/Clients eigene User mit Scope PRINT anlegen.

Token werden serverseitig nur als SHA-256-Hash gespeichert. Das Klartext-Token wird nur einmal bei der Erstellung zurückgegeben.

REST: Benutzer/Token anlegen

```
curl -X POST http://localhost:8080/api/auth/users \
-H "Content-Type: application/json" \
-d @src/main/resources/examples/rest-create-auth-user-request.json
```

Beispiel:

```
{
  "userName": "line01",
  "displayName": "Linie 01",
  "tokenName": "line01-print-token",
  "enabled": true,
  "scopes": ["PRINT", "PREVIEW"]
}
```

Antwort enthält das Token einmalig:

```
{
  "userName": "line01",
  "tokenName": "line01-print-token",
  "token": "NUR_EINMAL_SICHTBAR"
}
```

REST: Auth einschalten

```
curl -X PUT http://localhost:8080/api/auth/config \
  -H "Content-Type: application/json" \
  -H "X-Printservice-Token: ADMIN_TOKEN" \
  -d @src/main/resources/examples/rest-update-auth-config-enabled-request.json
```

REST: Drucken mit Token

```
curl -X POST http://localhost:8080/api/print \
  -H "Content-Type: application/json" \
  -H "X-Printservice-Token: YOUR_PRINT_TOKEN" \
  -d @src/main/resources/examples/rest-print-raw-tcp-zpl-request.json
```

Alternativ:

Authorization: Bearer YOUR_PRINT_TOKEN

Druckwarteschlange / Druckhistorie / Redruck

Jeder echte Druckjob wird unter `queue/print-jobs.json` gespeichert. Gespeichert werden u. a. Job-ID, Benutzer, Drucker, Vorlage, Render-Modus, Status und der Original-PrintRequest. Dadurch kann der Job später erneut gedruckt werden.

Eigene Jobs abrufen

```
curl "http://localhost:8080/api/queue?limit=100" \
  -H "X-Printservice-Token: YOUR_PRINT_TOKEN"
```

Wenn Auth aktiv ist, sieht ein normaler Benutzer nur seine eigenen Jobs.

Vollständige Liste abrufen

```
curl "http://localhost:8080/api/queue?full=true&limit=500" \
  -H "X-Printservice-Token: YOUR_ADMIN_TOKEN"
```

`full=true` benötigt Scope ADMIN oder ALL.

Nach User/Status/Vorlage filtern

```
curl "http://localhost:8080/api/queue?full=true&user=line01&status=SUBMITTED&templateName=s" \
  -H "X-Printservice-Token: YOUR_ADMIN_TOKEN"
```

Einzelnen Job abrufen

```
curl http://localhost:8080/api/queue/JOB_ID \  
-H "X-Printservice-Token: YOUR_PRINT_TOKEN"
```

Redruck

```
curl -X POST http://localhost:8080/api/queue/JOB_ID/reprint \  
-H "Content-Type: application/json" \  
-H "X-Printservice-Token: YOUR_PRINT_TOKEN" \  
-d @src/main/resources/examples/rest-reprint-request.json
```

Beispiel:

```
{  
  "copies": 1,  
  "printerAlias": "LABEL_LINE_01",  
  "dataOverride": {  
    "reprintReason": "Etikett beschädigt"  
  }  
}
```

Ohne `printerAlias` und ohne `dataOverride` wird der Originaljob unverändert erneut gesendet.

Druckerarten

WINDOWS_SPOOLER

Für PDF, HTML->PDF, PDF_TEMPLATE und normale Office-/A4-Drucker.

```
{  
  "alias": "OFFICE_A4",  
  "type": "WINDOWS_SPOOLER",  
  "systemName": "HP_LaserJet_A4",  
  "description": "A4 Drucker über Windows Spooler"  
}
```

RAW_TCP

Für schnelle Etikettendrucker mit ZPL/EPL/ESC-POS/PCL.

```
{  
  "alias": "LABEL_LINE_01",  
  "type": "RAW_TCP",  
  "host": "192.168.10.50",  
  "port": 9100,  
  "language": "ZPL",  
  "description": "Zebra Etikettendrucker direkt per TCP"  
}
```

REST-Endpunkte

GET /api/diagnostics/system
GET /api/printers/diagnostics
GET /api/printers/scan
GET /api/printers
POST /api/printers
DELETE /api/printers/{alias}
GET /api/templates/storage
GET /api/templates
GET /api/templates/{name}
POST /api/templates
PUT /api/templates/{name}
DELETE /api/templates/{name}
GET /api/barcodes/formats
GET /api/auth/status
GET /api/auth/me
GET /api/auth/users
POST /api/auth/users
DELETE /api/auth/users/{idOrUserName}
GET /api/auth/config
PUT /api/auth/config
POST /api/auth/reload
POST /api/print/preview
POST /api/print/preview/html
POST /api/print/rendered
POST /api/print
GET /api/docx/converters
POST /api/docx/validate
GET /api/queue
GET /api/queue/{jobId}
POST /api/queue/{jobId}/reprint

Render-Modi

Modus	Bedeutung
AUTO	Modus wird anhand des Vorlagentyps ermittelt
RAW	Text/ZPL/EPL direkt ausgeben
PDF	HTML nach PDF rendern
PDF_TEMPLATE	PDF-Vorlage mit Koordinatenfeldern befüllen
DOCX	DOCX befüllen und als DOCX bereitstellen

Barcode-Platzhalter in HTML/ZPL

```



```

ZPL direkt:

```
^F040,145^BCN,80,Y,N,N^FD{{serialNumber}}^FS
^F040,260^BQN,2,6^FDLA,{{serialNumber}}^FS
```

HTML-Tabelle

```
<table>
  <tbody>
    {{#each:items}}
      <tr>
        <td>{{text:pos}}</td>
        <td>{{text:partNumber}}</td>
        <td>{{text:description}}</td>
        <td>{{text:quantity}}</td>
        <td></td>
      </tr>
    {{/each}}
  </tbody>
</table>
```

PDF_TEMPLATE-Tabelle

```
{
  "name": "positionsTable",
  "type": "TABLE",
  "source": "items",
  "page": 1,
  "x": 18,
  "y": 78,
  "width": 174,
  "height": 125,
  "rowHeight": 13,
  "headerHeight": 8,
  "maxRows": 9,
  "overflow": "CUT",
  "fontSize": 7,
  "border": true,
  "columns": [
    { "title": "Pos.", "field": "pos", "width": 12, "align": "CENTER" },
    { "title": "Artikel", "field": "partNumber", "width": 29 },
  ]
}
```

```

    { "title": "Barcode", "type": "BARCODE", "barcodeType": "CODE_128", "field": "serialNum
  ]
}

```

DOCX -> PDF -> Druck ohne LibreOffice: DOCX4J_FO

Ab v2.3 gibt es zusätzlich zum externen LibreOffice-Weg den Java-only-Konverter:

```

{
  "renderMode": "DOCX",
  "docxConverter": "DOCX4J_FO"
}

```

Ablauf:

DOCX-Vorlage

- > Platzhalter/Barcode/QR im DOCX füllen
- > docx4j-export-fo wandelt DOCX nach XSL-FO
- > Apache FOP erzeugt PDF
- > PDFBox druckt das PDF

Vorteile:

- keine LibreOffice-Installation notwendig
- läuft komplett in der Java-Anwendung
- gut für einfache DOCX-Vorlagen mit Text, einfachen Tabellen, Bildern und Kopf-/Fußzeilen
- geeignet als optionaler, serverinterner Konverter

Grenzen:

- Layouttreue ist nicht so hoch wie bei Word/LibreOffice
- komplexe Word-Layouts, Shapes, Textfelder, SmartArt, exakte Seitenumbrüche und komplizierte Tabellen müssen pro Vorlage geprüft werden
- für Produktionsdruck ist PDF_TEMPLATE oder HTML -> PDF weiterhin robuster und meist schneller

Konverterstatus abrufen

```
curl http://localhost:8080/api/docx/converters
```

DOCX4J_FO mit einer Vorlage validieren

```

curl -X POST "http://localhost:8080/api/docx/validate?converter=DOCX4J_FO" \
-H "Content-Type: application/json" \
-H "X-Printservice-Token: YOUR_TOKEN" \
-d @src/main/resources/examples/rest-validate-docx4j-fo-request.json

```

Beispielrequest:

```
{
  "printerAlias": "OFFICE_A4",
  "templateName": "serienbegleitschein_docx",
  "renderMode": "DOCX",
  "docxConverter": "DOCX4J_FO",
  "includeRenderedBase64": true,
  "data": {
    "serialNumber": "SNR000001",
    "partNumber": "A123456",
    "workorderNumber": "WO-100200"
  }
}
```

Die Validierung prüft technisch, ob DOCX -> PDF erfolgreich erzeugt werden kann. Die Layout-Qualität muss anschließend visuell anhand des erzeugten PDFs geprüft werden.

DOCX mit DOCX4J_FO drucken

```
curl -X POST http://localhost:8080/api/print \
-H "Content-Type: application/json" \
-H "X-Printservice-Token: YOUR_TOKEN" \
-d @src/main/resources/examples/rest-print-docx4j-fo-request.json
```

```
{
  "printerAlias": "OFFICE_A4",
  "templateName": "serienbegleitschein_docx",
  "renderMode": "DOCX",
  "docxConverter": "DOCX4J_FO",
  "copies": 1,
  "pdfDpi": 300,
  "data": {
    "serialNumber": "SNR000001",
    "partNumber": "A123456",
    "workorderNumber": "WO-100200"
  }
}
```

Empfehlung: DOCX4J_FO als optionale Alternative einbauen, aber pro Vorlage in der Druckfreigabe validieren. Für feste Formulare bleibt PDF_TEMPLATE; für dynamische Tabellen bleibt HTML -> PDF; für Etiketten bleibt RAW_TCP/ZPL die erste Wahl.

Produktionsempfehlungen

- RAW_TCP für ZPL/EPL-Etikettendrucker verwenden.
- PDF_TEMPLATE für feste A4-Formulare verwenden.
- HTML -> PDF für flexible Tabellenformulare verwenden.

- Authentifizierung mindestens für `/api/print` aktivieren.
- Admin-Funktionen nach Initialeinrichtung schützen.
- Für Maschinen/Stationen separate Tokens mit eigenem User verwenden.
- `C:\printservice` regelmäßig sichern, besonders `templates`, `printers.json`, `auth.json` und `queue/print-jobs.json`.

HTML→PDF: robuste XHTML-Normalisierung ab v2.3

Der HTML→PDF-Druckpfad normalisiert HTML-Vorlagen jetzt vor dem PDF-Renderer automatisch zu XHTML. Hintergrund: `openhtmltopdf` ist deutlich strenger als ein Browser. Normales Browser-HTML wie `<br>`, `<img>`, fehlende `</td>` oder ein UTF-8-BOM vor `<html>` kann sonst mit einem XML-Fehler abbrechen, zum Beispiel:

Markup im Dokument vor dem Root-Element muss ordnungsgemäß formatiert sein.

Der Service führt deshalb vor dem Rendern aus:

```
HTML-Vorlage + JSON-Daten
→ Platzhalter und Tabellenblöcke rendern
→ BOM/ungültige Steuerzeichen entfernen
→ HTML-Fragmente bei Bedarf in <html><body>...</body></html> einbetten
→ Browser-HTML mit jsoup zu XHTML normalisieren
→ openhtmltopdf → PDF
→ Druck oder Download
```

Zusätzlich gibt es einen Validierungsendpunkt:

POST `/api/html/validate`

Beispiel:

```
curl -X POST http://localhost:8080/api/html/validate \
  -H "Content-Type: application/json" \
  -H "X-Printservice-Token: YOUR_TOKEN" \
  -d @src/main/resources/examples/rest-validate-html-pdf-request.json
```

Eine erfolgreiche Antwort enthält unter anderem:

```
{
  "ok": true,
  "templateName": "lieferschein_html_table",
  "renderMode": "PDF",
  "message": "HTML was rendered and normalized to XHTML successfully. It can be passed to op
}
```

Wenn die Vorlage versehentlich JSON oder PDF-Inhalt enthält, gibt der Service nun eine verständlichere Fehlermeldung aus, statt nur den SAX/XML-Fehler des Renderers weiterzugeben.

v2.4: Lieferschein-Demos und HTML->PDF Fehlerbehebung

HTML->PDF Fehler “Markup im Dokument vor dem Root-Element”

Der HTML->PDF Pfad normalisiert HTML jetzt vor dem openhtmltopdf Renderer. Dadurch werden typische Ursachen abgefangen: UTF-8 BOM, HTML-Fragmente ohne Root-Element, nicht XML-konforme Tags wie `<br>`, `<hr>` oder `<img>` ohne schließenden Slash.

Validierung vor dem Druck:

```
curl -X POST http://localhost:8080/api/html/validate \
-H "Content-Type: application/json" \
-H "X-Printservice-Token: YOUR_TOKEN" \
-d @src/main/resources/examples/rest-validate-html-pdf-request.json
```

Einfache PDF-Lieferscheinvorlage

```
config-example/templates/lieferschein_pdf_simple/
  template.json
  document.pdf
  fields.json
  sample-data.json
  print-request.json
```

DOCX-Lieferscheinvorlage

```
config-example/templates/lieferschein_docx/
  template.json
  document.docx
  sample-data.json
  print-request.json
  validate-request.json
```

Barcode-/QR-Platzhalter in DOCX müssen alleine in einem Absatz stehen:

```
{{barcode:CODE_128:serialNumber:360x90}}
{{qr:serialNumber}}
```

Für Druck ohne LibreOffice verwendet das Beispiel `docxConverter=DOCX4J_F0`. Jede DOCX-Vorlage sollte vor produktiver Nutzung validiert werden.