perplexity

End-to-End Guide to Clone iLovePDF & PDF Candy Tools Section

To replicate every working tool from iLovePDF and PDF Candy (≈90 utilities in your own website's **Tools** page, follow this comprehensive, step-by-step blueprint—from tech choices through deployment. Paste this prompt into Replit AI (or any AI assistant) to generate production-ready code.

1. Technology Stack

Frontend

- Next.js 14 + React 18 (SSR/SSG, routing)
- Tailwind CSS 3.4 (dark theme, utility-first)
- Shadcn/UI (accessible components)
- Three.js + @react-three/fiber + @react-three/drei (3D cards & backgrounds)
- Framer Motion & GSAP 3 (UI/scroll animations)
- Lottie-React (micro-interactions)

Backend
– **Python Flask 2** (heavy PDF/OCR/AI services)

- PyMuPDF (fitz) – merge/split/compress/encrypt/decrypt PDFs[1] [2]
- pdf2docx + python-docx + python-pptx (Office conversions)
- pytesseract + OpenCV (OCR/scan)[3]
- Camelot-py + pandas (table extraction)
  – **Node.js 20 + Express 5** (image, utility, scraping)
- Sharp (crop/compress/format conversions) over Jimp for performance[4] [5]
- qrcode (QR generator)
- yt-dl-wrapped (YouTube thumbnail)
- Puppeteer (Instagram DP)
- Prettier (code/text formatter)

Infrastructure

- Redis 7 (4-min link tokens + rate-limit)
- Nginx (X-Accel-Redirect, HTTPS)
- Docker-Compose (frontend, python-api, node-api, redis, nginx)

2. Tools Inventory & Routing

Implement **all** tools from both sites—grouped with REST endpoints:

Category	Tools	Endpoint Pattern
PDF⇆Office	PDF→Word, Word→PDF, PDF→Excel, Excel→PDF, PDF→PowerPoint, PPT→PDF	POST `/api/pdf/office/{tool}`
PDF⇆Image	PDF→JPG/PNG/TIFF, JPG/PNG→PDF (batch)	POST `/api/pdf/image/{tool}`
eBook⇆PDF	EPUB/MOBI/FB2/ODT/DjVu/TXT→PDF; PDF→EPUB	POST `/api/pdf/ebook/{tool}`
Core PDF Ops	Merge, Split (pages/range), Compress (fast/quality), Rotate, Crop, Organize, Delete…	POST `/api/pdf/core/{tool}`
OCR & AI	Scan→OCR, Extract Tables, Summarize Text/PDF, AI Grammar Check	POST `/api/pdf/ai/{tool}`
Metadata & Validation	Edit Metadata, Validate PDF/A	POST `/api/pdf/meta/{tool}`
Utility Extras	HTML→PDF, Sign PDF, QR Code, TTS, Word/Page Counter, Code Formatter…	POST `/api/util/{tool}`

Each returns JSON `{ downloadUrl, expiresIn:240 }`.

3. Secure File Flow

1. **Upload** → `POST /upload` (multer in Node or Flask-Uploads in Python)
2. **Validate** mime-type, signature, size ≤ 50 MB; virus-scan via ClamAV[?].
3. **Store** in `/tmp/uploads/{uuid4}/`; perms 600; record in Redis:

```
SETEX download:{token} 240 '{"path":"/tmp/.../file.ext"}'
```

4. **Process** by appropriate microservice (Python or Node).
5. **Generate link**: `/download/{token}`. Redis TTL = 240s.
6. **Cleanup**: Cron job or Redis expiry triggers directory deletion.

4. 3D-Animated UI Design

Theme variables

```
:root {
  --bg: #0a0a0a; --card: #121212; --glass: rgba(255,255,255,0.08);
  --accent1: #00f5ff; --accent2: #8b5cf6; --accent3: #10b981;
}
```

Components

- **ToolCard**: Three.js “Float” box, glassmorphism, hover tilt, ripple on click.

- **UploadZone**: Drag-drop with particle.js trail, real-time validation feedback.
- **ProgressRing**: 3D ring with colored arcs + % label.
- **Grid**: Masonry flex → mobile single-column; infinite scroll.

Interactions

- GSAP ScrollTrigger for card entry.
- Framer Motion for button/overlay animations.
- Lottie for success/error micro-animations.

5. Frontend Code Structure

```
/frontend
├─ /components
│   ├─ /ui         # Shadcn/UI wrappers
│   ├─ /3d         # Three.js scenes (ToolCard, Background)
│   ├─ /tools      # Per-tool pages & forms
│   └─ /layout     # Navbar, Footer, Auth
├─ /pages
│   ├─ index.js    # Tools grid
│   ├─ /tool/[slug].js # Individual tool UI
│   └─ /api        # Next.js API proxies (optional)
├─ tailwind.config.js
└─ next.config.js
```

6. Backend Code Structure

```
/backend-python
├─ app.py            # Flask app & blueprint registration
├─ /services
│   ├─ pdf_service.py  # Core PDF ops via PyMuPDF & PyPDF2
│   ├─ ocr_service.py  # pytesseract + OpenCV
│   ├─ ai_service.py   # Transformers summarizer, LanguageTool
│   └─ ebook_service.py # pdf2docx & python-pptx
├─ /utils
│   ├─ security.py    # validation/sanitization
│   ├─ storage.py     # upload/download helpers
│   └─ cleanup.py     # scheduled file deletion

/backend-node
├─ server.js         # Express app
├─ /routes
│   ├─ image.js      # Sharp-based endpoints
│   ├─ util.js       # QR, thumbnail, formatter
│   └─ upload.js     # multer config + Redis link gen
├─ /middleware
│   ├─ security.js   # mime/signature checks
│   └─ rateLimit.js  # express-rate-limit

/docker-compose.yml
```

7. Deployment & Verification

1. **Docker-Compose** up; ensure `frontend`, `python-backend`, `node-backend`, `redis`, `nginx` services.
2. **Env Vars**: `REDIS_URL`, `JWT_SECRET`, `MAX_UPLOAD_MB`.
3. **Reverse Proxy**: Nginx routes `/api/pdf/*` to Flask (5000), `/api/util/*` to Node (5001).
4. **HTTPS**: Let's Encrypt via Certbot.
5. **Health Checks**: Ping all endpoints; verify no 404/500.
6. **Functional Tests**: Upload large PDF (25 MB), compress, merge; download within 4 min; confirm file auto-deleted.
7. **Performance Audits**: Lighthouse Performance ≥ 90; Accessibility ≥ 95.

Paste this entire plan into Replit AI. It will scaffold the **exact**, production-quality codebase—no placeholders—to replicate **all** tools in a stunning, dark, 3D-animated interface.

⁂

1. https://www.educative.io/courses/pdf-management-python/walkthrough-top-python-libraries-for-pdf-processing
2. https://pythonology.eu/what-is-the-best-python-pdf-library/
3. https://dev.to/mhamzap10/5-python-pdf-conversion-packages-for-document-management-2m0o
4. https://npm-compare.com/image-js,jimp,sharp
5. https://npm-compare.com/canvas,imagescript,jimp,sharp