—

PDF Signature — Software Design Document (SDD)

Project: PDF Signature (SignFlow)
Version: 0.1.0
Tech Stack: Python 3.11, Reflex 0.8.24.post1, FastAPI (via Reflex API), PyMuPDF (fitz), TailwindCSS, Vanilla JS + signature_pad
Repository Root: `https://github.com/milochen0418/pdf_signature/`

—

Table of Contents

—

1. Overview

PDF Signature is a lightweight web application that lets a user:

1. Upload a PDF
2. Preview and navigate pages (server-renders pages to PNG for viewing)

3. Draw one or more “signature boxes” on a page
4. Open a signature pad modal to draw a signature
5. Embed the signature content into the PDF permanently
6. Export and download the signed PDF

The app is implemented using **Reflex** (Python-only web app framework). State is managed in `PDFState`, which coordinates upload handling, page rendering, signature box placement, signature capture, and PDF export.

—

2. Goals and Non-goals

Goals

- Simple guided workflow: upload → place signature → export
- Reliable signed output (signature permanently embedded into PDF)
- Fast preview and page navigation
- Minimal UI complexity

Non-goals

- Cryptographic / PKI digital signatures (X.509, PAdES, certificate-based signing)
- Multi-user collaboration
- Persistent document storage / user accounts
- Advanced annotation tools (text, stamps, highlights)

—

3. User Experience and Core Flows

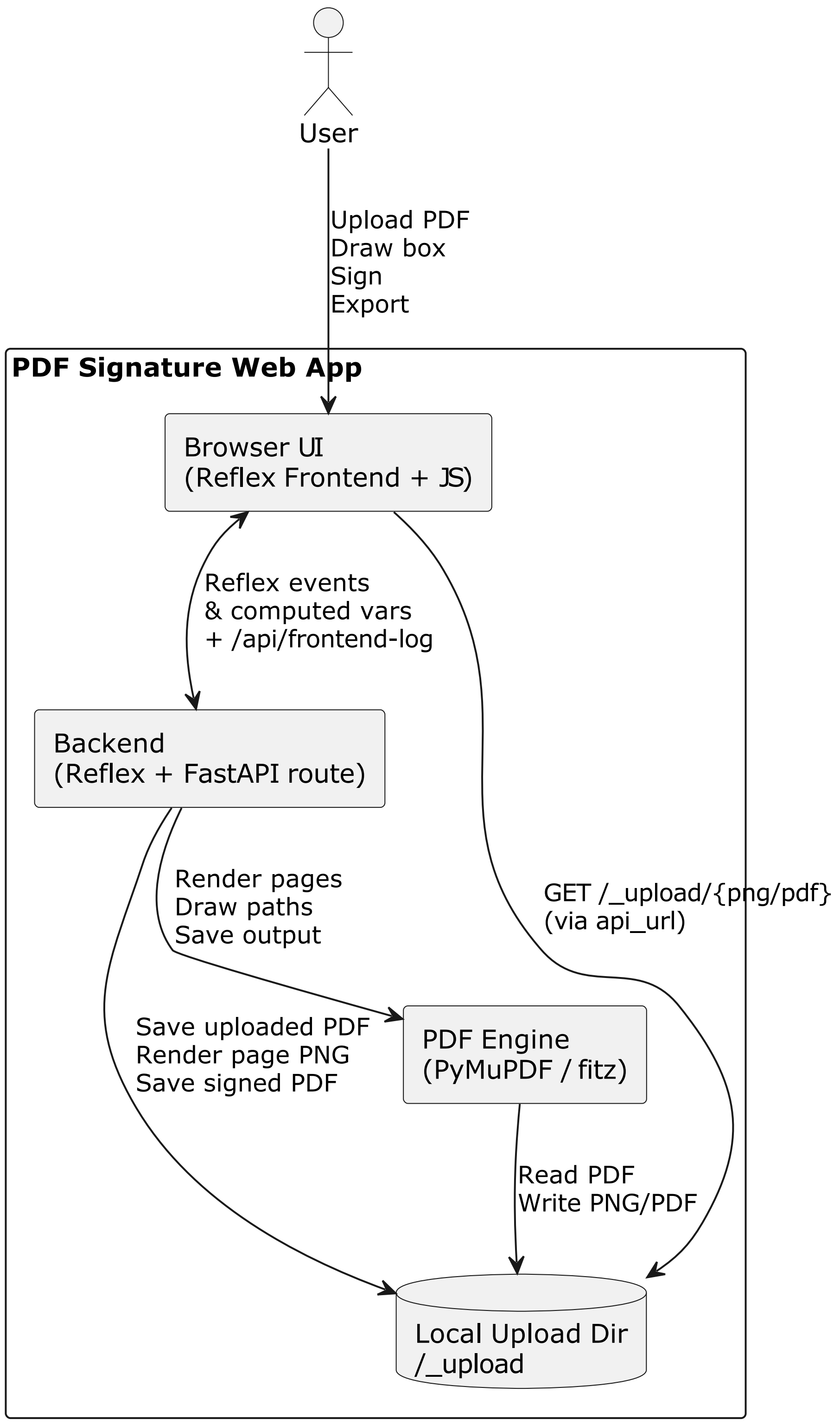
Primary Flow

1. Upload a PDF via sidebar dropzone
2. View page image in main viewer
3. Click “Draw Box” and drag to create a signature area
4. Click the signature box to open signature modal
5. Draw signature and click “Apply Signature”
6. Click “Export PDF”
7. Download signed PDF from sidebar

—

4. System Architecture

High-level view (Context)



Key Architectural Choice

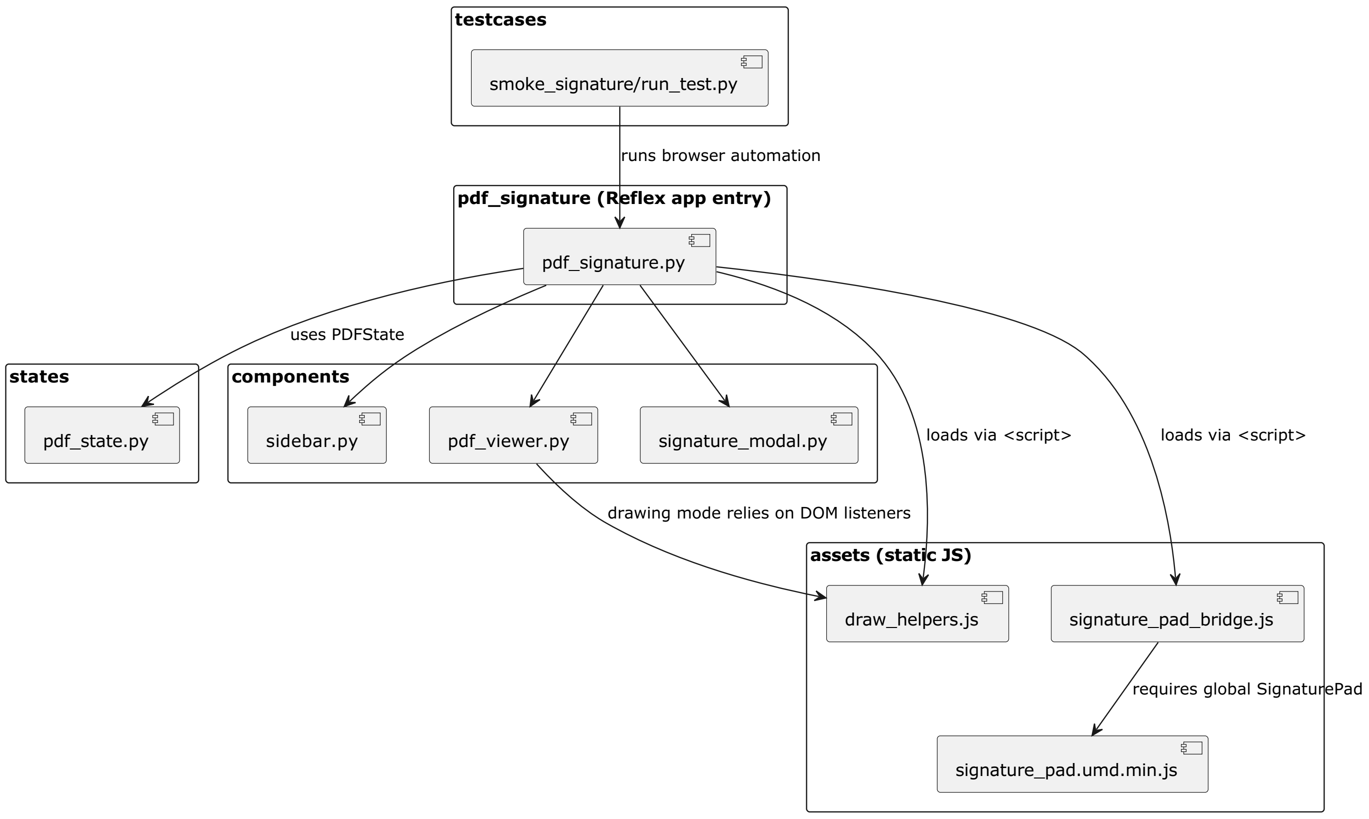
Instead of rendering PDF pages via PDF.js in browser, this app **renders the current page on the server** (PyMuPDF) into a PNG (`*_page{n}.png`) and displays it as an `<img>`. Signature boxes are overlays positioned in **percentage coordinates**, so they scale with zoom.

5. Source Tree and Module Responsibilities

Directory Structure (abridged)

```
pdf_signature/
  README.md
  rxconfig.py
  pyproject.toml
  assets/
    signature_pad.umd.min.js
    signature_pad_bridge.js
    draw_helpers.js
  pdf_signature/
    pdf_signature.py
    states/
      pdf_state.py
    components/
      sidebar.py
      pdf_viewer.py
      signature_modal.py
  testcases/
    smoke_signature/run_test.py
  docs/images/*.png
```

Module Map



6. Data Model

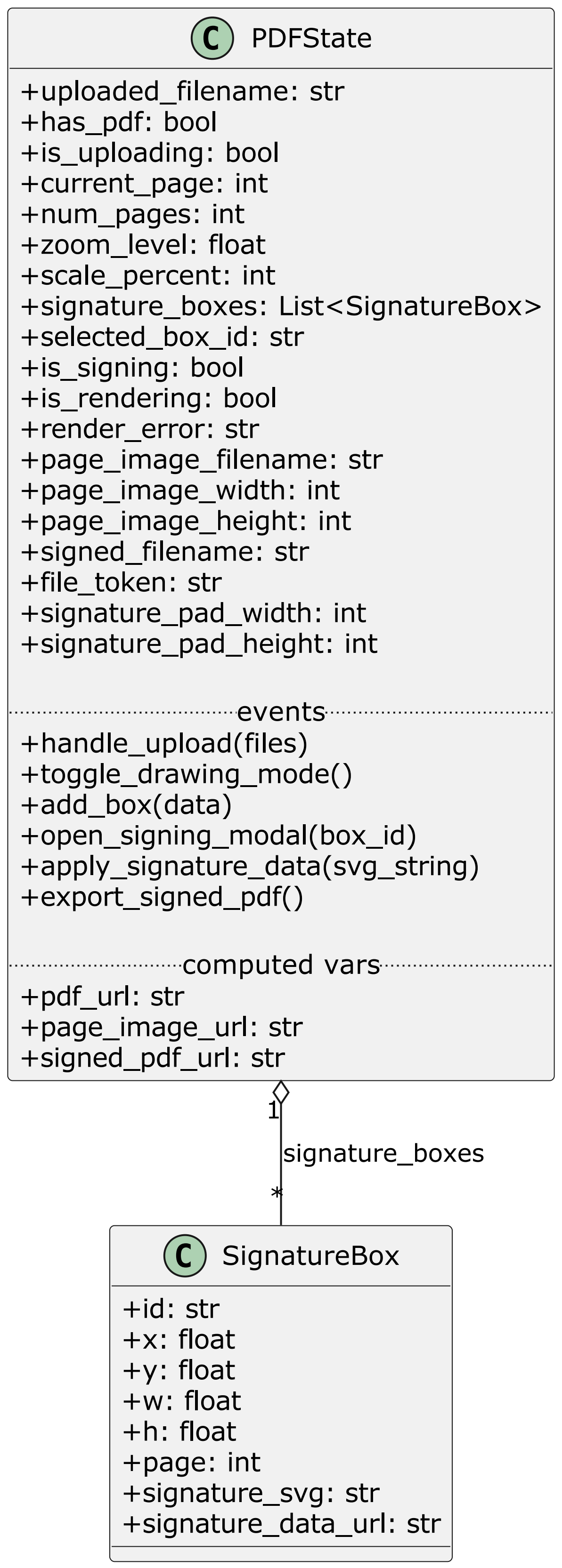
SignatureBox (in-memory)

`SignatureBox` is stored in `PDFState.signature_boxes` and uses **relative percent coordinates** to position boxes on a rendered page image.

Fields (from `SignatureBox` TypedDict):

- `id: str`
- `x, y, w, h: float` (percent, relative to displayed page image container)
- `page: int` (1-based)
- `signature_svg: str` (raw SVG from signature_pad)
- `signature_data_url: str` (base64 svg data URL for preview inside the box)

Data Model Diagram



—

7. Backend Design (State + Rendering + Export)

7.1 Upload Handling

- UI uses `rx.upload.root(...)` (sidebar) with `max_files=1`, accept PDF MIME/extension.
- `PDFState.handle_upload(files)`:
- Validates `.pdf` extension
- Saves file into `rx.get_upload_dir()` with randomized prefix
- Generates a `file_token`
- Resets relevant state (boxes, pages, errors)
- Renders page 1 to PNG via `_render_page_image(1, file_path)`

7.2 Server-side Page Rendering

`_render_page_image(page_index, file_path)`:

- Opens PDF with `fitz.open(file_path)`
- Loads page (`page_index - 1`)
- Renders pixmap at 2x scale: `page.get_pixmap(matrix=fitz.Matrix(2,2))`
- Saves PNG into upload directory: `{file_token}_page{page_index}.png`
- Stores image filename and pixel dimensions into state

7.3 Export Signed PDF

`export_signed_pdf()`:

- Opens original PDF
- Iterates over all `signature_boxes`
- For each box with non-empty `signature_svg`:
- Maps `% coordinates → PDF page coordinates` using `page.rect.width/height`
- Parses signature SVG into:
  - Bezier segments from `<path ... d="M ... C ...">`
  - Dots from `<circle ...>`
  - Draws shapes using PyMuPDF `page.new_shape()`:
  - `draw_bezier(...)` + `finish(color=..., width=..., lineCap=1, lineJoin=1)`
  - `draw_circle(...)` + `finish(fill=...)`
  - Commits shape onto page

    - Saves output to `{file_token}_signed.pdf`
    - Makes download link available (sidebar renders anchor if `signed_pdf_url != ""`)

—

8. Frontend Design (Reflex UI + JS Bridges)

8.1 UI Composition

`pdf_signature/pdf_signature.py` builds the app layout:

- Loads JS assets via `<script src="http://localhost:8000/...">`
- `/signature_pad.umd.min.js`
- `/signature_pad_bridge.js`
- `/draw_helpers.js`
- Renders:
- `signature_modal()`
- Sidebar (`sidebar()`)
- Main area:
    - If `PDFState.has_pdf`: `pdf_controls()` + `pdf_viewer_canvas()`
    - Else: empty-state prompt

8.2 Signature Capture (signature_pad)

- `signature_modal.py` displays a modal with a `<canvas id="signature-canvas">`.
- `PDFState.open_signing_modal(box_id)`:
- sets `selected_box_id`, `is_signing=True`
- executes JS `initSignaturePad('signature-canvas')` after a short timeout
- "Apply Signature" button:
- calls JS `window.getSignatureSVG()`
- uses callback to `PDFState.apply_signature_data(svg_string)`
- `apply_signature_data`:
- stores raw SVG
- builds a preview `data:image/svg+xml;base64,...`
- updates the selected SignatureBox

- closes modal

8.3 Drawing Signature Boxes (draw_helpers.js)

- User toggles draw mode via “Draw Box” button → `PDFState.toggle_drawing_mode()`.
- When `is_drawing_box=True`, the viewer container gets crosshair cursor class.
- `draw_helpers.js` attaches document-level listeners once:

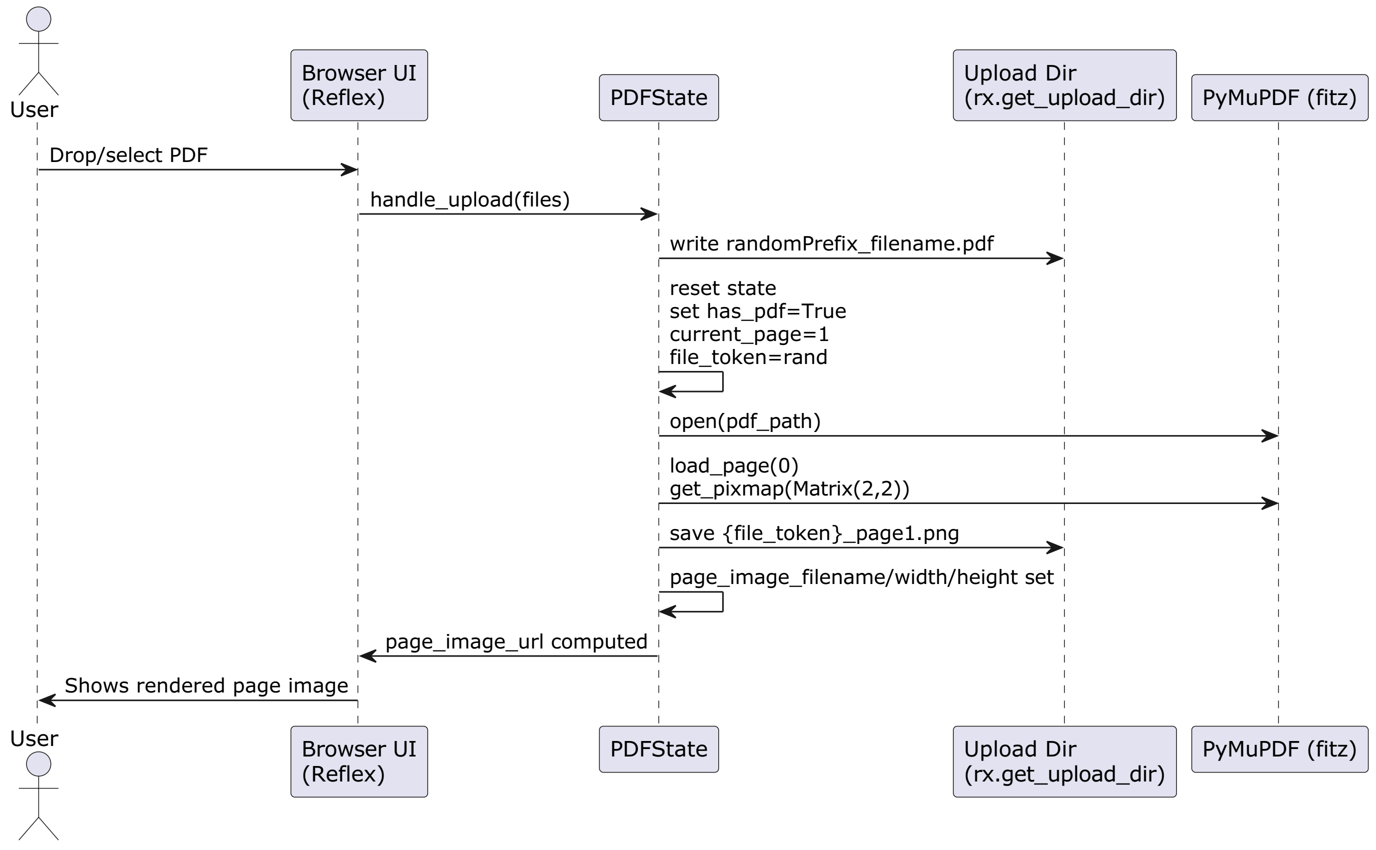
- On mousedown in container: creates a temporary fixed-position div showing a dashed rectangle

- On mousemove: adjusts temp rectangle
- On mouseup:
    - Computes relative `%` coordinates (box rect / container rect)
    - Writes JSON into hidden `<input id="new-box-data-input">`
    - Dispatches `input` and `change` events so Reflex triggers `PDFState.add_box(data)`

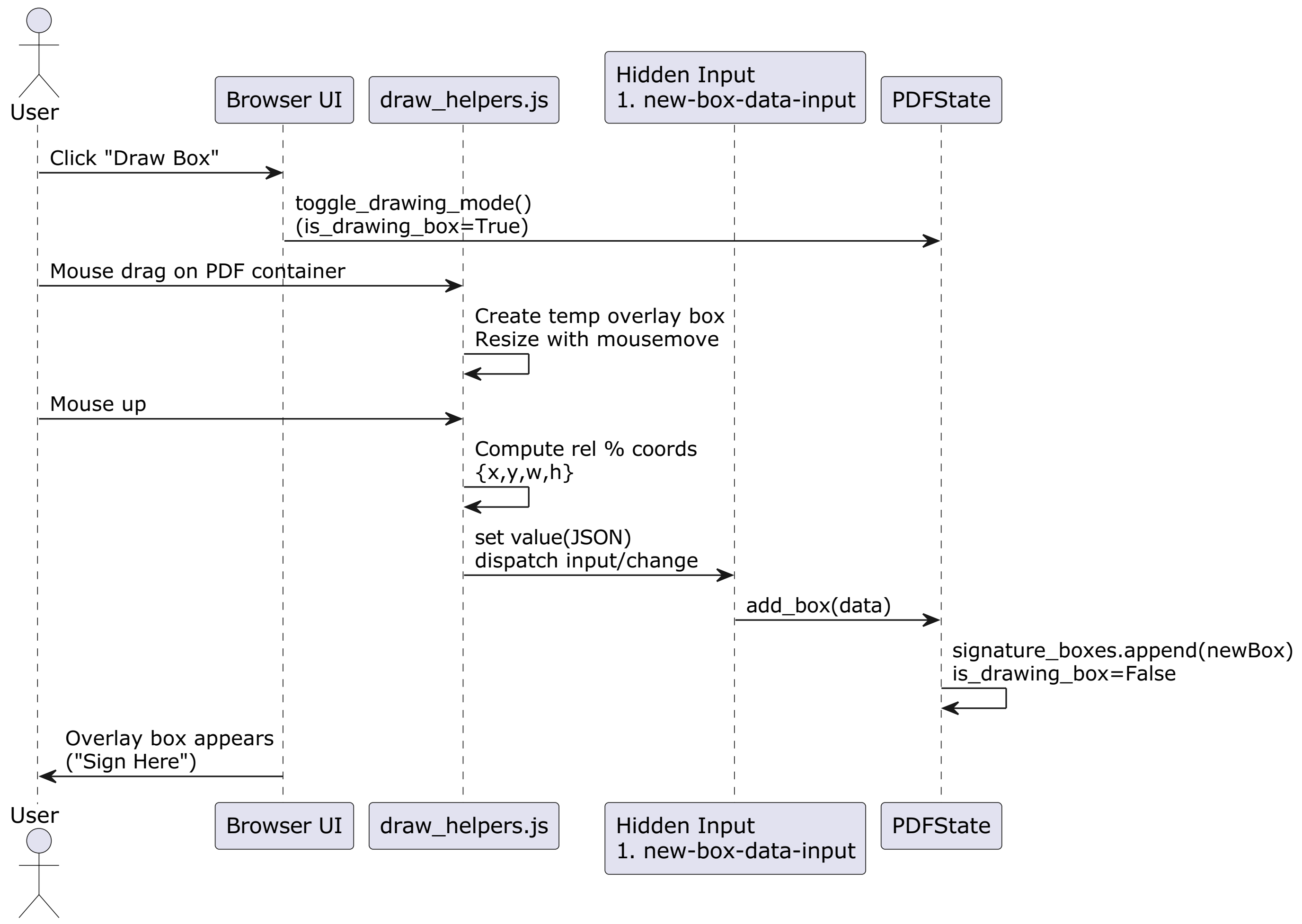
—

9. Detailed Workflows (Sequence Diagrams)

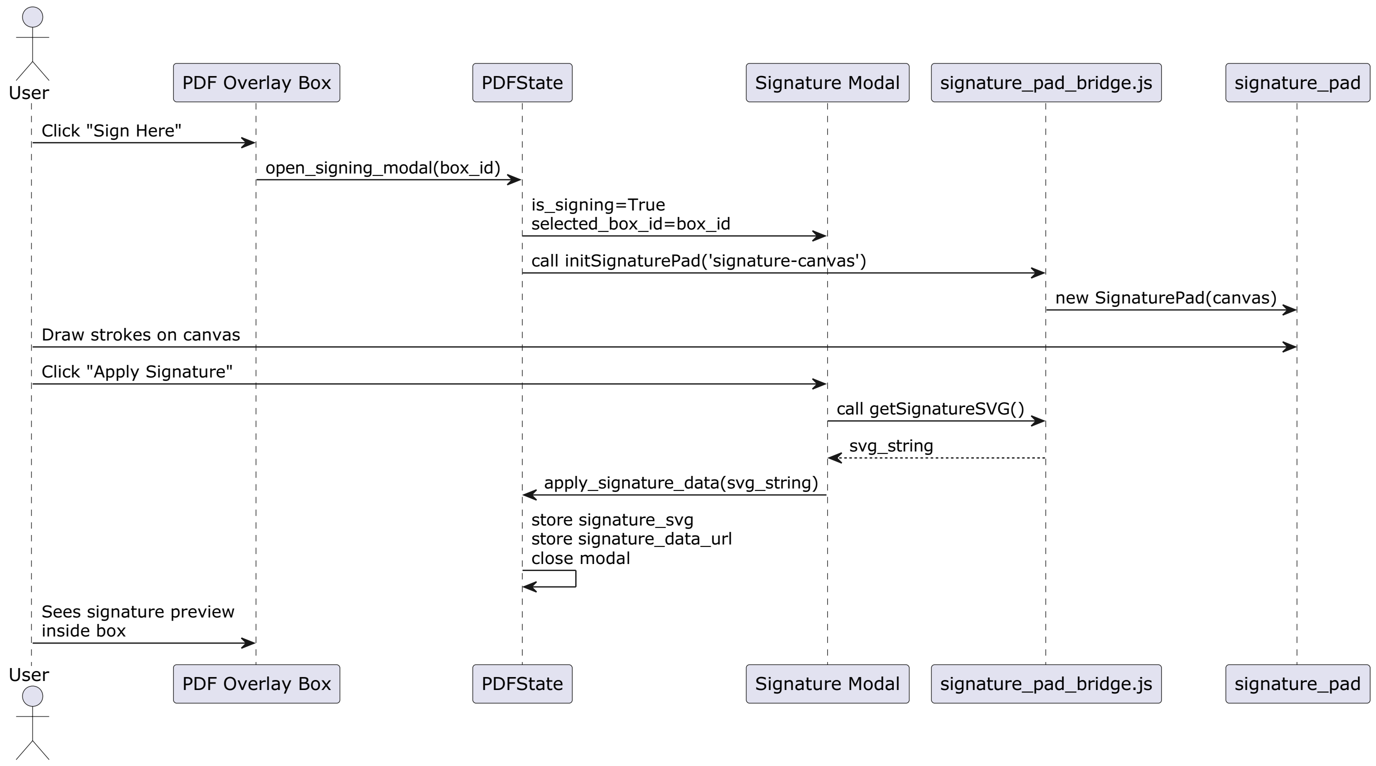
9.1 Upload → Render First Page



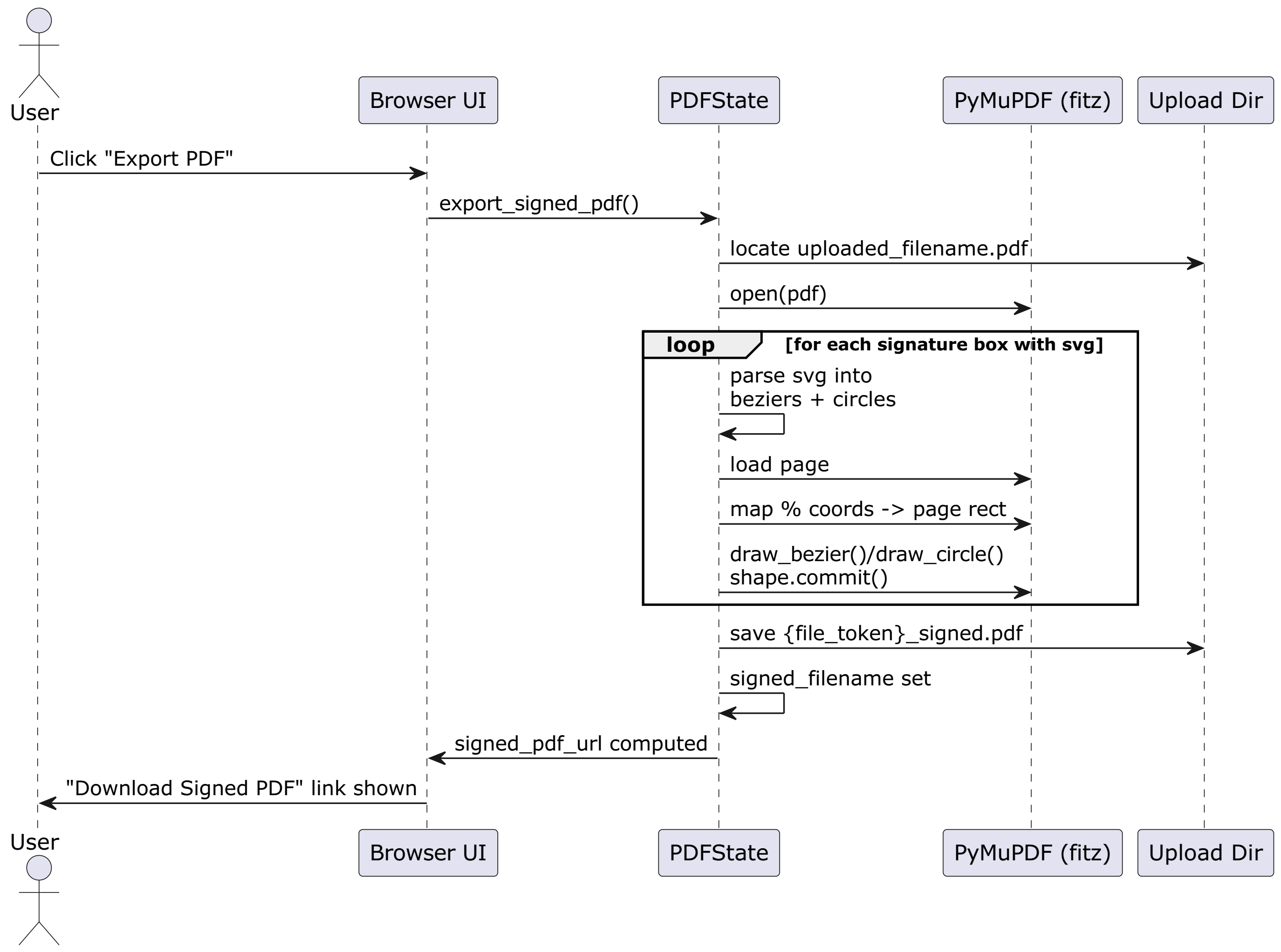
9.2 Draw Signature Box



9.3 Sign in Modal → Overlay Preview



9.4 Export Signed PDF


User
Browser UI
PDFState
PyMuPDF (fitz)
Upload Dir
Click "Export PDF"
export_signed_pdf()
locate uploaded_filename.pdf
open(pdf)
loop
[for each signature box with svg]
parse svg into beziers + circles
load page
map % coords -> page rect
draw_bezier()/draw_circle() shape.commit()
save {file_token}_signed.pdf
signed_filename set
signed_pdf_url computed
"Download Signed PDF" link shown

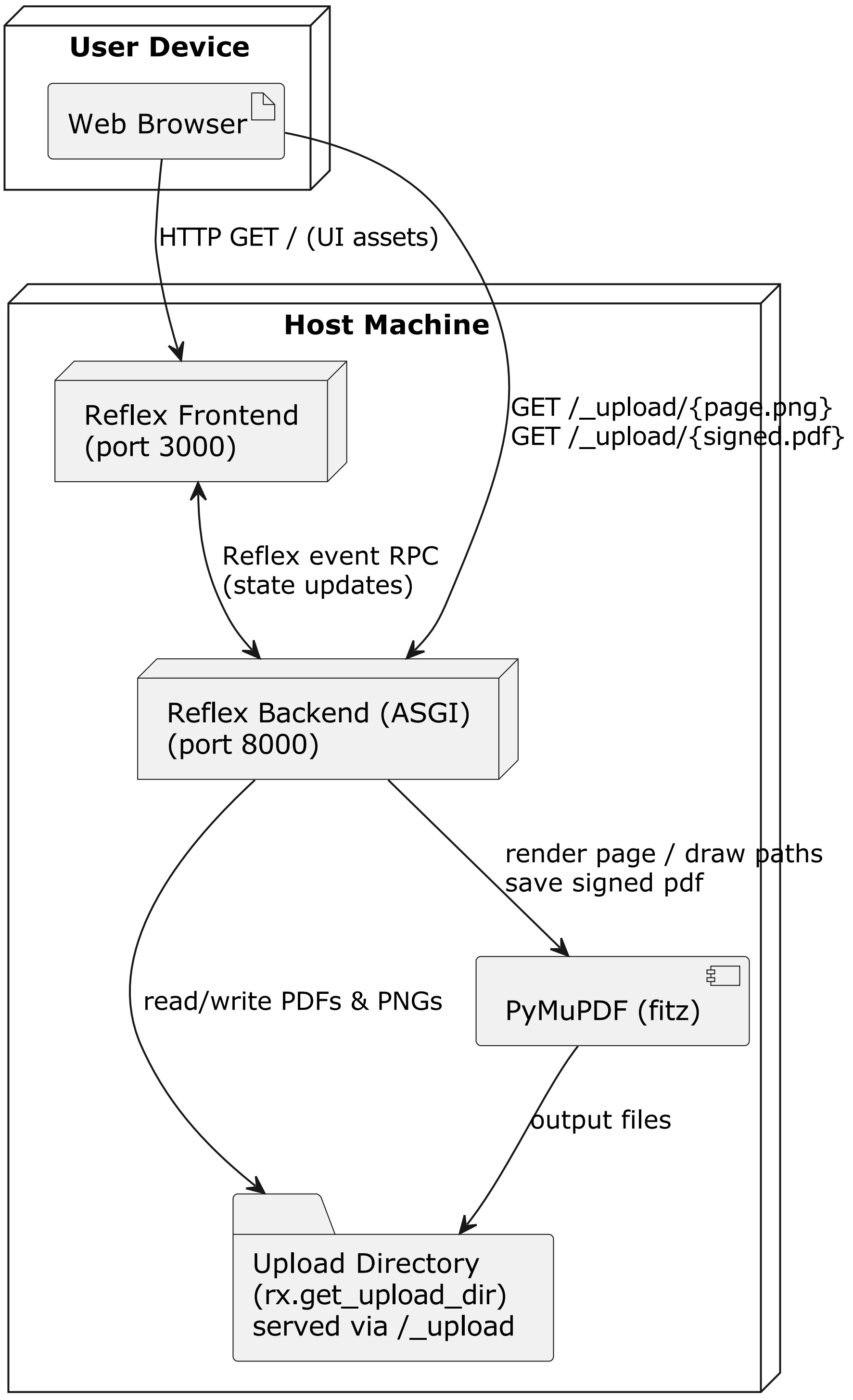
—

10. Deployment

This app typically runs as a Reflex dev server:

- Frontend: `http://localhost:3000`
- Backend API: `http://localhost:8000`
- Uploaded artifacts served via backend `/_upload/...`

Deployment Diagram



—

11. Error Handling and Observability

Error Surfaces

- `PDFState.render_error` displayed as overlay in viewer if non-empty
- Upload validation: rejects non-PDF extension
- Export: “Original PDF not found” if upload missing

Frontend Console Log Bridge

`pdf_signature.py` injects a script that forwards `console.log/warn/error`, `window.onerror`, and `unhandledrejection` to:

- `POST /api/frontend-log`

Backend route:

- `frontend_log(request: Request)` logs messages under logger `"frontend"`.

This gives basic client-side observability from server logs.

—

12. Security Considerations

- **File type validation** currently checks `.pdf` filename extension, not MIME sniffing.
- Recommendation: validate magic header `%PDF-` server-side.
- Uploaded files stored in a shared upload directory and served by predictable URLs.
- Recommendation: enforce per-file random tokens in URLs (already partially via randomized filenames).
- No authentication/authorization.
- Intended as single-user/local tool; deploying publicly would require auth and storage controls.
- SVG parsing is done via regex and only supports expected signature_pad output patterns.
- Recommendation: sanitize/limit SVG size and complexity; reject unexpected tags.

—

13. Performance Considerations

- Rendering each page uses `Matrix(2,2)` which can be heavy for large PDFs.
- Potential improvements:
    - Cache rendered pages per `file_token` and `page_index` (already saved as PNG; reuse if exists)
    - Render at adaptive resolution based on zoom or viewport

        - Storage cleanup: PNGs and signed PDFs can accumulate in upload directory.
- Add cleanup policy (time-based purge) or “Clear Document” action.

–

14. Testing Strategy

E2E (Playwright)

`testcases/smoke_signature/run_test.py` automates:

1. Upload PDF
2. Draw box
3. Open modal
4. Draw signature strokes
5. Apply signature (preview appears)
6. Export PDF (download link appears)
7. (Optional) validate output PDF content using PyMuPDF

This is the primary regression safety net for the end-to-end signing workflow.

Recommended Additional Tests

- Unit tests for:
- `_parse_signature_svg()` with representative SVG samples
- `% coord → PDF rect` mapping correctness
- Export behavior across multiple pages and boxes

–

15. Future Improvements

Product / UX

- Multi-page thumbnails navigation
- Multiple documents session history (in-memory)
- Drag-to-move / resize signature boxes after creation
- Support typed name signatures (text) alongside drawn signatures

Engineering

- Strong PDF validation (magic bytes)
- Signature parsing more robust (XML parser with strict allowlist)
- File cleanup / quota
- Optional persistence layer (DB) if multi-user is needed
- Add structured logs / trace IDs for frontend-log events

—

Appendix A — Key Files (Quick Reference)

- `pdf_signature/pdf_signature.py`: app entry, layout, JS injection, frontend log route
- `pdf_signature/states/pdf_state.py`: upload, render, box management, signature application, export
- `pdf_signature/components/pdf_viewer.py`: viewer UI, overlays, controls, hidden input bridge
- `pdf_signature/components/signature_modal.py`: signature canvas modal
- `assets/signature_pad_bridge.js`: init/clear/get SVG bridge for signature_pad
- `assets/draw_helpers.js`: rectangle drag-to-create signature box
- `testcases/smoke_signature/run_test.py`: Playwright E2E smoke test

—