■ PYDATAUI ENTERPRISE FRAMEWORK

PyDataUI Developer Cookbook

The Production Full-Stack Python Guide for Data Scientists, AI Engineers & Backend Teams

Framework Version: v1.5.0 (Latest Production Release)

Core Engine: Native FastAPI + Starlette + FastHTML + HTMX DOM Morphing

Target Surfaces: Web ASGI • Instant Public HTTPS Tunnels • Docker • Desktop • Mobile • TUI

Published on PyPI: https://pypi.org/project/pydataui/1.5.0/

Table of Contents

1. Architecture, Mental Model & 10-Second Quickstart	Streamlit DX vs Next.js SSR vs FastAPI backend
2. The Two Development Tracks	Track 1: Procedural DX vs Track 2: Declarative App Router
3. Advanced Layout Engine (v1.5.0)	Flexbox (pdu.flex), Responsive Grid (pdu.grid), Containers & Spacers
4. Coordinate Positioning & Docked Overlays	Relative & Absolute Positioning, Pinned Viewport Toolbars, Floating Badges
5. Navigation, Tabs & Headless Multi-Step Wizards	Expanded Tabs, Pill Segments, and Headless Wizard Sequences
6. Commercial Dashboard & Analytics Suite	Executive StatCards, Inline SVG Sparklines, Badges, Pricing Cards
7. Forms, Temporal Inputs & Interactive Controls	Date/Time Selectors, Search Bars, Batched Forms, Reactive Inputs
8. Production Authentication, Users & Role Protection	create_user, login, logout, require_auth, and @pdu.page(roles=[...])
9. FastAPI & Starlette Production Integration	Direct app.fastapi, CORS Middleware, Swagger UI Filtering, Background Tasks
10. The Theming Engine: 20 Presets & Custom CSS	Violet, Emerald, Zinc Dark, Custom Hex, and Palette Dictionaries
11. Multi-Platform Targets & Deployments	ASGI Web, Docker, Instant Tunnels (share=True), Capacitor Mobile, Desktop, TUI
12. Production Cookbook: 5 Complete Enterprise Applications	Full runnable code for AI Copilot, FinTech MRR, K8s Telemetry, Checkout Wizard
13. Architectural Trade-Offs & The 5 Limitations	Detailed comparison vs Next.js, FastAPI & Streamlit
Appendix: Complete API Reference & Cheatsheet	Exhaustive parameter breakdown for every function in pydataui

Chapter 1: Architecture, Mental Model & 10-Second Quickstart

Modern web development has historically forced a painful choice: developers could either use **Streamlit** for rapid 10-second prototyping (and suffer from top-to-bottom script re-runs, session bugs, and zero built-in REST API), or build with **Next.js + FastAPI** (gaining performance and production APIs, but paying with 450 MB of node_modules, dual TypeScript/Python languages, and schema glue code).

PyDataUI completely eliminates this trade-off. On the surface, you write procedural, top-to-bottom Python with zero classes or decorators. Underneath, an enterprise ASGI engine powered by **FastAPI, Starlette, FastHTML, and HTMX DOM morphing** executes surgically. When a user clicks a button, only the affected fragment updates in place without restarting Python execution from line 1.

KEY ARCHITECTURAL ADVANTAGE: 0ms In-Memory Model Access: Because your UI runs in the exact same Python process as your PyTorch, LangChain, or Pandas models, you never incur network serialization overhead between frontend and backend. Model weights and dataframes stay in RAM.

The 10-Second Quickstart (`app.py`)

Install with `pip install pydataui` and run the following script with `python app.py`:

```
import pydataui as pdu
import pandas as pd

# 1. Page Configuration & Theme
pdu.set_page_config(title="AI & Cluster Telemetry", layout="wide", theme="zinc", dark_mode=True)

# 2. Modern Sidebar Controls
with pdu.sidebar():
    pdu.title("Controls")
    pdu.theme_toggle()
    model = pdu.selectbox("Select Model", ["Llama-3.3-70B", "DeepSeek-R1", "GPT-4o"])
    temp = pdu.slider("Temperature", 0.0, 1.0, 0.7)

# 3. Header & Status
pdu.title("Production Cluster Telemetry")
pdu.success("All 16 H100 SXM5 nodes online. Telemetry streaming active.")

# 4. Multi-Column KPI Grid
col1, col2, col3 = pdu.columns([1, 1, 1])
with col1: pdu.metric("Active Sessions", "2,840", delta="+12.4%")
with col2: pdu.metric("P99 Inference Latency", "14.2ms", delta="-3.1ms")
with col3: pdu.metric("Token Throughput", "8.2k tokens/s", delta="+18.5%")

# 5. Interactive Tabs & Tables
tab_data, tab_chat = pdu.tabs(["Data Explorer", "Operations Copilot"])
with tab_data:
    df = pd.DataFrame({"Worker": ["node-01", "node-02"], "GPU Load": ["88%", "92%"]})
    pdu.write(df)
    pdu.line_chart([65, 72, 80, 85, 92, 88, 91, 95])

with tab_chat:
    pdu.chat_message("user").write("Are there any thermal anomalies?")
    pdu.chat_message("assistant").write("All nodes operating within normal thermal limits.")
    pdu.chat_input("Ask the operations assistant a question...")

if __name__ == "__main__":
    pdu.run(port=8000, share=True)  # Ready in 3.5s with instant public HTTPS link!
```

Execution Endpoints

Surface	URL / Access Point	Description
Local Web UI	http://127.0.0.1:8000	Server-Side Rendered dashboard with instant HTMX morphing.
Public Tunnel	https://*.app.lhr.life	Zero-config public HTTPS link generated in 3.5s via `share=True`.
FastAPI Swagger Docs	http://127.0.0.1:8000/docs	Interactive OpenAPI 3.1 Swagger UI documentation.

Chapter 2: The Two Development Tracks

PyDataUI accommodates two distinct engineering workflows depending on team requirements and project complexity:

Track 1: Procedural Streamlit-Style Prototyping

Ideal for data scientists, ML researchers, and rapid prototyping. You write top-to-bottom procedural code with `import pydataui as pdu`. There are no component classes, no decorators, and no state initialization boilerplate. Everything renders in sequential order.

```
import pydataui as pdu

pdu.set_page_config(title="Rapid Data Tool", theme="emerald")
pdu.title("Quick CSV Analyzer")

file = pdu.file_uploader("Upload CSV", accept=[".csv"])
if file:
    import pandas as pd
    df = pd.read_csv(file)
    pdu.write(df)
    pdu.bar_chart(df.mean(numeric_only=True))
```

Track 2: Declarative Next.js-Style Full-Stack SaaS

Ideal for production engineering teams building enterprise SaaS portals. PyDataUI provides a **Next.js App Router equivalent in pure Python** with Server Actions, typed Pydantic models, and auto-generated REST routes:

```
from pydataui import App, Container, Card, Heading, Text, Button, Table, Row, state
from pydantic import BaseModel, Field

app = App(title="CloudAI Enterprise SaaS")

# 1. Server State & Server Actions (Next.js "use server" equivalent)
s = state(credits=1500, deployments=[{"id": "dep-01", "model": "Llama-3.3", "status": "Running"}])

@s.action
def deploy_model(model_name: str = "Mistral-Large"):
    s.deployments.append({"id": f"dep-0{len(s.deployments) + 1}", "model": model_name, "status": "Running"})
    s.credits -= 100

# 2. Server-Rendered Component Page (app/page.tsx equivalent)
@app.page("/")
def dashboard_page():
    return Container(
        Row(
            Heading("Enterprise AI Dashboard", level=1),
            Button("+ Deploy Mistral", on_click=lambda: deploy_model("Mistral-Large"), variant="primary"),
            class_name="flex justify-between items-center mb-6"
        ),
        Card(
            Heading("Remaining Credits", level=4),
            Text(s.credits, class_name="text-2xl font-bold text-emerald-600"),
            Table(data=s.deployments),
            class_name="p-6 shadow-sm"
        ),
        class_name="max-w-5xl mx-auto p-8"
    )

# 3. Built-In FastAPI Route (Next.js route.ts equivalent)
class PromptRequest(BaseModel):
    prompt: str = Field(..., min_length=1)

@app.api("/api/v1/generate", method="POST", tags=["AI Inference"])
async def generate_api(body: PromptRequest):
    return {"success": True, "output": f"Completion for: {body.prompt[:40]}..."}

if __name__ == "__main__":
    app.run(port=8000)
```

Chapter 3: Advanced Layout Engine (v1.5.0)

PyDataUI v1.5.0 brings full Next.js/Tailwind layout expressiveness directly into procedural Python. Developers have full control over CSS Flexbox, Responsive CSS Grid, spacing, and structural containers without ever touching CSS files.

1. Flexbox Layout Engine (`pdu.flex`, `pdu.row`, `pdu.column`)

Control alignment, justification, gaps, and wrapping with both `with` blocks and direct child parameters:

```
import pydataui as pdu

# Horizontal navigation bar with Flexbox between-alignment
with pdu.flex(direction="row", justify="between", align="center", gap=4):
    pdu.subheader("Production Cluster Alpha")
    with pdu.row(align="center", gap=2):
        pdu.badge("US-EAST-1", color="zinc", variant="outline")
        pdu.badge("Operational", color="emerald", variant="subtle")
        pdu.button("+ Deploy Pod", type="primary")

# Vertical stack with stretch alignment
with pdu.flex(direction="column", align="stretch", gap=3):
    pdu.caption("Cluster Load Capacity")
    pdu.progress(85)
```

Parameter	Type	Default	Description & Options
direction	str	'row'	'row' or 'column'.
justify	str	'start'	'start', 'center', 'end', 'between', 'around', 'evenly'.
align	str	'stretch'	'stretch', 'start', 'center', 'end', 'baseline'.
gap	int | str	3	Gap unit (1 to 12) or exact CSS string ('16px').
wrap	bool | str	False	Whether items wrap onto multiple lines.

2. Responsive CSS Grid Engine (`pdu.grid`)

Build auto-wrapping, responsive dashboard grids without writing CSS media queries. The `min_column_width` prop automatically creates an auto-fit CSS grid (`repeat(auto-fit, minmax(min_width, 1fr))`):

```
# Auto-responsive card grid: columns wrap automatically when cards reach 260px!
with pdu.grid(min_column_width="260px", gap=4):
    pdu.stat_card("Active Nodes", "32 / 32", change="100% OK", sparkline=[10, 10, 10], color="emerald")
    pdu.stat_card("P99 Latency", "8.2 ms", change="-2.1ms", sparkline=[14, 12, 10, 8], color="indigo")
    pdu.stat_card("VRAM Allocated", "1,180 GB", change="+12%", sparkline=[60, 75, 85, 92], color="amber")
    pdu.stat_card("Token Throughput", "18.4k t/s", change="+24.8%", sparkline=[12, 15, 17, 18], color="rose")

# Fixed 3-column grid with custom row and column gaps:
with pdu.grid(columns=3, row_gap=3, column_gap=6):
    pdu.metric("Cluster CPU", "42%")
    pdu.metric("Memory", "68%")
    pdu.metric("Network I/O", "1.2 GB/s")
```

3. Spacing Primitives (`pdu.spacer`, `pdu.divider`)

```
pdu.spacer(height=24)        # 24px vertical spacing block
pdu.divider(label="OR")      # Horizontal rule with centered text label
```

Chapter 4: Coordinate Positioning & Docked Overlays

In enterprise interfaces, components frequently need to break out of normal flow: floating status badges in card corners, sticky action bars docked to the bottom of the viewport, and absolute modal overlays. In PyDataUI v1.5.0, **every single visual component supports first-class CSS coordinate positioning**.

First-Class Positioning Props

Prop	Type	Default	Description & Examples
position	str	None	'relative', 'absolute', 'fixed', or 'sticky'.
top	str | int	None	Top coordinate offset ('14px', '1rem', 0).
bottom	str | int	None	Bottom coordinate offset.
left	str | int	None	Left coordinate offset.
right	str | int	None	Right coordinate offset.
inset	str | int	None	Shorthand for all four edges (e.g. 0 for full fill).
z_index	int	None	Stacking order priority (e.g. 50).

Pattern 1: Floating Absolute Badge on Relative Card

By setting `position='relative'` on a `card` container, any child with `position='absolute'` anchors to the card's boundaries:

```
import pydataui as pdu

with pdu.card(position="relative"):
    # Floating Badge pinned 14px from top and right edges
    pdu.badge("HOTSPOT DETECTED", color="rose", position="absolute", top="14px", right="14px")
    pdu.title("Worker Node 04")
    pdu.caption("High thermal signature detected during distributed LLM checkpointing.")
    pdu.line_chart([65, 72, 80, 85, 94, 98, 82], color="rose")
```

Pattern 2: Viewport-Docked Sticky Action Bar (`pdu.pinned`)

`pdu.pinned(position='bottom')` docks a container to the viewport edge, remaining in view even while the user scrolls through thousands of rows of data:

```
# Docked viewport action bar at bottom of window
with pdu.pinned(position="bottom"):
    with pdu.flex(justify="between", align="center", class_name="max-w-4xl mx-auto py-2"):
        pdu.text("■■ 3 worker nodes require kernel updates before next training run.")
        with pdu.row(gap=2):
            pdu.button("Ignore for 24h", type="outline")
            pdu.button("Apply Batch Update", type="primary")
```

Chapter 5: Navigation, Tabs & Headless Multi-Step Wizards

Tabs in PyDataUI operate as high-speed in-memory client-side switchers. In v1.5.0, `pdu.tabs` was significantly expanded with three major capabilities: equal width expansion, segmented pill variants, and headless multi-step wizard mode.

Tab Modes & Configuration

```
import pydataui as pdu

# 1. Full-Width Equal Distribution Tabs
# Distributes tab buttons evenly across 100% of the available container width
tab_telemetry, tab_security = pdu.tabs(["System Telemetry", "Security Audit"], expand=True)

# 2. Modern Segmented Pill Tabs
# Renders a sleek pill switcher (ideal for date filters, chart ranges, or toggle views)
tab_week, tab_month = pdu.tabs(["7 Days", "30 Days"], variant="pill")
```

3. Headless Multi-Step Wizard (`hide_tab_buttons=True`)

By hiding the navigation buttons, `pdu.tabs` transforms into a clean, state-driven multi-step wizard. You can build multi-stage checkouts, onboarding flows, and AI interview forms in pure Python:

```
import pydataui as pdu

pdu.set_page_config(title="SaaS Onboarding Wizard", theme="violet")

if "step" not in pdu.session_state:
    pdu.session_state["step"] = 0

# Multi-Step Wizard: Tab headers completely hidden from user!
steps = pdu.tabs(["Account", "Organization", "Billing", "Finish"], hide_tab_buttons=True)

# Step 0: Account Details
with steps[0]:
    with pdu.card(title="Step 1: Account Information"):
        pdu.text_input("Full Name", key="wiz_name")
        pdu.text_input("Work Email", key="wiz_email")
        pdu.button("Next: Organization →", type="primary", on_click=lambda: pdu.session_state.update({"step": 1}))

# Step 1: Organization Details
with steps[1]:
    with pdu.card(title="Step 2: Organization Setup"):
        pdu.text_input("Company Name", key="wiz_company")
        pdu.selectbox("Team Size", ["1-10", "11-50", "51-200", "200+"])
        with pdu.row(justify="between"):
            pdu.button("← Back", on_click=lambda: pdu.session_state.update({"step": 0}))
            pdu.button("Next: Billing →", type="primary", on_click=lambda: pdu.session_state.update({"step": 2}))

# Step 2: Plan Selection
with steps[2]:
    with pdu.card(title="Step 3: Select Plan"):
        with pdu.grid(columns=2, gap=4):
            pdu.pricing_card("Pro", "$49", features=["Up to 10 nodes"])
            pdu.pricing_card("Enterprise", "$199", features=["Dedicated VPC"], highlighted=True)
        with pdu.row(justify="between"):
            pdu.button("← Back", on_click=lambda: pdu.session_state.update({"step": 1}))
            pdu.button("Complete Setup ■", type="primary", on_click=lambda: pdu.session_state.update({"step": 3}))

# Step 3: Success Screen
with steps[3]:
    pdu.success("■ Welcome aboard! Your cluster workspace is provisioned.")
```

Chapter 6: Commercial Dashboard & Analytics Suite

Commercial executive dashboards require high-density visual telemetry: KPI metrics with delta trends, mini trendlines (sparklines), status tags, and conversion pricing cards. PyDataUI provides pure zero-dependency primitives for all of them.

1. Executive StatCards with Inline SVG Sparklines

`pdu.stat_card` renders an executive-grade metric card with an icon, delta trend pill, and an integrated inline SVG sparkline graph that renders in ~0.001ms without any external JavaScript chart libraries:

```
import pydataui as pdu

with pdu.grid(columns=4, gap=4):
    pdu.stat_card("Annual Run Rate", "$1,840,500", change="+24.8%", sparkline=[90, 110, 130, 145, 160, 184], icon="■", color
    pdu.stat_card("Monthly Recurring", "$154,200", change="+18.2%", sparkline=[92, 104, 118, 132, 145, 154], icon="■", color
    pdu.stat_card("Active Accounts", "2,840", change="+142 mo", sparkline=[2200, 2400, 2600, 2840], icon="■", color="indigo"
    pdu.stat_card("Net Retention", "118.4%", change="TOP 5%", sparkline=[112, 114, 116, 118], icon="■", color="amber")
```

2. Pure SVG Inline Sparklines (`pdu.sparkline`)

Use `pdu.sparkline` anywhere in tables, cards, or custom layouts:

```
pdu.sparkline([12, 14, 18, 15, 22, 28, 35, 32, 42], color="indigo", width=140, height=36, stroke_width=2, fill=True)
```

3. Semantic Status Badges (`pdu.badge`)

Badges support three visual variants (`'subtle'`, `'solid'`, `'outline'`) and any of the 20 theme color presets or custom HEX values:

```
pdu.badge("Operational", color="emerald", variant="subtle")
pdu.badge("Critical Failure", color="rose", variant="solid")
pdu.badge("Draft Release", color="zinc", variant="outline")
pdu.badge("Custom Hex", color="#7c3aed", variant="solid")
```

4. SaaS Pricing Cards (`pdu.pricing_card`)

```
with pdu.grid(columns=3, gap=6):
    pdu.pricing_card("Starter", "$29", features=["Up to 5 nodes", "Community Slack"])
    pdu.pricing_card("Pro", "$149", features=["50 nodes", "24/7 SLA", "90-day logs"], highlighted=True, badge_label="BEST VA
    pdu.pricing_card("Enterprise", "Custom", features=["Unlimited nodes", "Dedicated VPC"])
```

Chapter 7: Forms, Temporal Inputs & Interactive Controls

PyDataUI provides a complete catalog of interactive form inputs, temporal pickers, and instant HTMX search bars bound directly to session state.

Temporal & Search Input Primitives

```
import pydataui as pdu

# 1. Native HTML5 Date Picker bound to session_state
pdu.date_input("Audit Window Date", value="2026-11-18", key="audit_date")

# 2. Native HTML5 Time Picker bound to session_state
pdu.time_input("Scheduled Maintenance Time", value="02:30", key="maint_time")

# 3. Real-time Search Input with Instant HTMX Trigger
def filter_logs(query):
    print("Executing real-time search for:", query)

pdu.search_input(placeholder="Search telemetry logs...", key="log_query", on_search=filter_logs)
```

Batched Forms (`pdu.form`, `pdu.form_submit_button`)

By default, input changes can trigger reactive events. When editing complex records or user profiles, wrap inputs in a `pdu.form`. Input state updates are batched together and dispatched only when the submit button is clicked:

```
with pdu.card(title="Edit Cluster Configuration"):
    with pdu.form("cluster_form"):
        pdu.text_input("Cluster Name", value="us-east-prod", key="cfg_name")
        pdu.number_input("Max Replicas", min_value=1, max_value=64, value=16, key="cfg_replicas")
        pdu.slider("Inference Timeout (s)", min_value=5, max_value=120, value=30, key="cfg_timeout")

        def save_config():
            name = pdu.session_state.get("cfg_name")
            replicas = pdu.session_state.get("cfg_replicas")
            pdu.success(f"Saved {name} with {replicas} replicas!")

        pdu.form_submit_button("Save Configuration", on_click=save_config)
```

Complete Input Component Catalog

Component	Signature Summary	Key Functionality
`pdu.text_input`	label, value='', key=None, type='text'	Single line text or password input.
`pdu.number_input`	label, min_value=None, max_value=None, step=1	Numeric stepper with range clamps.
`pdu.slider`	label, min_value=0, max_value=100, step=1	Visual range slider bar.
`pdu.selectbox`	label, options, index=0, key=None	Dropdown selection picker.
`pdu.multiselect`	label, options, default=None	Multi-tag picker.
`pdu.checkbox`	label, value=False, key=None	Boolean toggle checkbox.
`pdu.toggle`	label, value=False, key=None	iOS-style switch toggle.
`pdu.file_uploader`	label, accept=None, multiple=False	Drag & drop file upload zone.
`pdu.color_picker`	label, value='#4f46e5'	Hex color picker.

Chapter 8: Production Authentication, Users & Role Protection

PyDataUI includes a built-in enterprise authentication manager supporting SQLite, PostgreSQL, or in-memory persistence. You can register users, verify passwords with cryptographic hashing, protect multi-page routes with Role-Based Access Control (RBAC), and guard Swagger UI.

```
import pydataui as pdu

pdu.set_page_config(title="SaaS Portal", theme="indigo", dark_mode=True)

# 1. Public Registration Page
@pdu.page("/signup", public=True)
def signup_screen():
    with pdu.card(title="Create Account"):
        with pdu.form("signup_form"):
            pdu.text_input("Username", key="reg_user")
            pdu.text_input("Password", type="password", key="reg_pass")

            def handle_signup():
                u, p = pdu.session_state.get("reg_user"), pdu.session_state.get("reg_pass")
                if pdu.create_user(u, p, roles=["admin"]):
                    pdu.login(u, p)
                    pdu.switch_page("/dashboard")
                else:
                    pdu.error("User already exists")

            pdu.form_submit_button("Sign Up", on_click=handle_signup)

# 2. Public Login Page
@pdu.page("/login", public=True)
def login_screen():
    with pdu.card(title="Sign In"):
        with pdu.form("login_form"):
            pdu.text_input("Username", key="login_user")
            pdu.text_input("Password", type="password", key="login_pass")

            def handle_login():
                u, p = pdu.session_state.get("login_user"), pdu.session_state.get("login_pass")
                if pdu.login(u, p):
                    pdu.switch_page("/dashboard")
                else:
                    pdu.error("Invalid credentials")

            pdu.form_submit_button("Sign In", on_click=handle_login)

# 3. Role-Protected Dashboard (Requires "admin" role)
@pdu.page("/dashboard", roles=["admin"])
def dashboard_screen():
    with pdu.sidebar():
        pdu.write(f"Logged in as: **{pdu.session_state.get('user')}**")
        pdu.button("Log Out", on_click=lambda: (pdu.logout(), pdu.switch_page("/login")))

    pdu.title("Executive Dashboard")
    pdu.metric("Cluster Health", "100% OK")

if __name__ == "__main__":
    pdu.run(port=8000)
```

Authentication API Reference

Function	Signature	Description
`pdu.create_user`	username, password, email=None, roles=None	Registers new user with hashed credentials.
`pdu.login`	username, password	Authenticates user and writes token to session cookie.
`pdu.logout`		Terminates active user session.
`pdu.require_auth`	roles=None, redirect_url='/login'	Route guard. Halts rendering and redirects if unauthenticated.
@pdu.page	path, roles=['admin'], public=False	Page decorator with built-in RBAC role protection.

Chapter 9: FastAPI & Starlette Production Integration

Because PyDataUI runs natively on FastAPI and Starlette, **100% of the FastAPI ecosystem is available out of the box**: APIRouter, BackgroundTasks, middlewares (CORS, GZip), dependency injection, and Swagger documentation hardening.

1. Direct FastAPI Engine Access (`pdu.get_fastapi()` / `app.fastapi`)

```
import pydataui as pdu
from fastapi import APIRouter, BackgroundTasks

# Retrieve underlying FastAPI application instance
fastapi_app = pdu.get_fastapi()  # or app.fastapi

# Attach native FastAPI APIRouter
router = APIRouter(prefix="/v1")

def async_worker(event: str):
    print(f"[WORKER] Background event processed: {event}")

@router.post("/trigger-scan")
def trigger_scan(bg: BackgroundTasks):
    bg.add_task(async_worker, "Cluster scan enqueued")
    return {"status": "enqueued"}

fastapi_app.include_router(router)
```

2. Starlette & FastAPI Middlewares (CORS, Security)

Attach standard middlewares directly via `fastapi_app.add_middleware` or `app.add_middleware`:

```
from starlette.middleware.cors import CORSMiddleware
import pydataui as pdu

fastapi_app = pdu.get_fastapi()
fastapi_app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://mycompany.com"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)
```

3. Production Swagger UI Hardening

When moving to production, you have complete control over OpenAPI documentation:

```
import pydataui as pdu

# Option A: Completely remove /docs and /openapi.json in production (returns HTTP 404)
pdu.set_page_config(docs_url=None, openapi_url=None)

# Option B: Filter and customize Swagger UI parameters
pdu.set_page_config(
    swagger_ui_parameters={
        "filter": True,                          # Enable instant search filter in Swagger
        "defaultModelsExpandDepth": -1,          # Hide data models by default
        "syntaxHighlight.theme": "monokai",      # Monokai dark syntax highlighting
        "persistAuthorization": True             # Remember auth tokens across refreshes
    }
)

# Option C: Lock down /docs behind Role-Based Authentication
# Only authenticated users with the "admin" role can access the Swagger UI!
pdu.set_page_config(docs_roles=["admin"])
```

Chapter 10: The Theming Engine: 20 Presets & Custom CSS

PyDataUI includes **20 official out-of-the-box color presets** matching all standard Shadcn/Tailwind design tokens. Every theme supports light mode, high-contrast dark mode, and custom HEX brand palettes in 1 line of Python.

Preset Name	Category	Hex Primary	Ideal Application Domain
violet	Blues & Purples	#7C3AED	AI Agent Copilots, LLM Workbenches, Chatbots
emerald	Greens & Cyans	#059669	FinTech, SaaS Billing, Revenue Analytics
zinc	Modern Neutrals	#18181B	DevOps, Kubernetes Clusters, GPU Telemetry
indigo	Blues & Purples	#4F46E5	Default Enterprise SaaS, Admin Portals
amber	Pinks & Warm	#D97706	Logistics, Supply Chain, Warehouse Fleet
rose	Pinks & Warm	#E11D48	Incident Management, Alerting, Security Operations
slate	Modern Neutrals	#0F172A	Executive Dashboards, Business Intelligence
cyan	Greens & Cyans	#0891B2	Cloud Infrastructure, Storage Telemetry
blue	Blues & Purples	#2563EB	Corporate Portals, Developer Documentation
stone	Modern Neutrals	#1C1917	Publishing, Content Management Systems

Activating Themes & Custom Brand Palettes

```
import pydataui as pdu

# 1. Built-in Preset with Dark Mode
pdu.set_page_config(theme="zinc", dark_mode=True)

# 2. Custom Brand Hex
pdu.set_page_config(theme="#4f46e5")

# 3. Custom Brand Dictionary Palette
pdu.set_page_config(theme={
    "primary": "#6366f1",
    "primary_hover": "#4f46e5",
    "bg": "#0f172a",
    "accent": "#a855f7"
})

# Render interactive theme toggle anywhere in the app
pdu.theme_toggle()
```

Chapter 11: Multi-Platform Targets & Deployments

PyDataUI keeps its web core ultra-lean (~295 KB wheel package). When required, optional extensions enable native packaging across mobile, desktop, and remote terminal environments with zero bloat.

Target Surface	Technology Used	Command / Invocation	Package Footprint
Web ASGI	Uvicorn / Gunicorn	python app.py	Pure Python (<300 KB)
Instant Public Link	SSH Reverse Tunnel	`pdu.run(share=True)`	0 MB (3.5s link)
Docker Container	python:3.11-slim	docker build -t app .	~120 MB image
Mobile (iOS / Android)	Capacitor 6	pydataui mobile build	Standard APK / IPA
Desktop (.app / .exe)	Tauri / pywebview	pydataui desktop build	~15 MB (Zero Electron)
Terminal UI (TUI)	Textual / ANSI	pydataui run --tui app.py	Zero extra deps

Production Dockerfile

```
FROM python:3.11-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000
CMD ["pydataui", "run", "app.py", "--host", "0.0.0.0", "--port", "8000", "--workers", "4"]
```

Instant 3.5s Zero-Config Public HTTPS Tunnel

```
# One parameter gives an instant public HTTPS tunnel for stakeholders:
if __name__ == "__main__":
    pdu.run(port=8000, share=True)
# Output:
# ■ Local Web UI: http://127.0.0.1:8000
# ■ Public Tunnel: https://a1d60ba6d715e7.app.lhr.life
```

Chapter 12: Production Cookbook Recipes

Recipe 1: Enterprise AI Copilot & Agent Playground (`theme='violet'`)

Features collapsible reasoning thought traces, temperature controls, and multi-turn chat streaming:

```
import pydataui as pdu

pdu.set_page_config(title="DeepAgent Copilot", theme="violet", dark_mode=True)

with pdu.sidebar():
    pdu.title("Agent Settings")
    model = pdu.selectbox("Inference Engine", ["DeepSeek-R1-Distill-70B", "Llama-3.3-70B"])
    temp = pdu.slider("Temperature", 0.0, 1.0, 0.70)
    pdu.checkbox("Stream Reasoning Traces", value=True)

pdu.title("Distributed Agent Workbench")

with pdu.chat_message("user"):
    pdu.write("Analyze the P99 latency spike on cluster node-04 and remediate.")

with pdu.chat_message("assistant"):
    with pdu.expander("■ Thought Process: 1,420 tokens analyzed (0.4s)"):
        pdu.write("Inspected Prometheus traces. Found cache compaction lock on worker-04.")
    pdu.write("Remediation executed: Pods re-routed to node-02. Current P99 latency: **12.1ms**.")

pdu.chat_input("Ask Copilot or type /command...")

if __name__ == "__main__":
    pdu.run(port=8000)
```

Recipe 2: FinTech Executive MRR Acceleration Dashboard (`theme='emerald'`)

Renders ARR/MRR velocity, customer cohorts, and pure inline SVG sparklines in ~0.001ms:

```
import pydataui as pdu
import pandas as pd

pdu.set_page_config(title="ApexPay FinTech Analytics", layout="wide", theme="emerald")

pdu.title("ApexPay Revenue Operations & Cohorts")
pdu.success("Stripe Webhook Sync: Active • 99.99% Gateway Uptime")

# Executive Stat Cards with inline Sparklines
with pdu.grid(columns=4, gap=4):
    pdu.stat_card("Annual Run Rate (ARR)", "$1,840,500", change="+24.8%", sparkline=[90, 110, 130, 145, 160, 184], icon="■", color="emerald")
    pdu.stat_card("Monthly Recurring (MRR)", "$154,200", change="+18.2%", sparkline=[92, 104, 118, 132, 145, 154], icon="■", color="emerald")
    pdu.stat_card("Active Accounts", "2,840", change="+142 mo", sparkline=[2200, 2400, 2600, 2840], icon="■", color="indigo")
    pdu.stat_card("Net Retention Rate", "118.4%", change="TOP 5%", sparkline=[112, 114, 116, 118], icon="■", color="amber")

pdu.spacer(height=16)

with pdu.flex(direction="row", justify="between", align="center"):
    pdu.subheader("Revenue Ingestion Ledger")
    pdu.date_input("Audit Window", value="2026-11-18", key="ledger_date")

df = pd.DataFrame({
    "Transaction ID": [f"tx-894{i}" for i in range(1, 6)],
    "Customer": ["Acme Corp", "BioHealth AI", "FinFlow Inc", "DataMesh Labs", "ScaleOps"],
    "Tier": ["Enterprise", "Pro", "Pro", "Enterprise", "Pro"],
    "Amount": ["$12,400", "$4,900", "$4,900", "$15,200", "$4,900"],
    "Status": ["Settled", "Settled", "Settled", "Processing", "Settled"]
})
pdu.write(df)

if __name__ == "__main__":
    pdu.run(port=8000)
```

Recipe 3: High-Density Kubernetes GPU Cluster Telemetry (`theme='zinc'`)

Low-eye-strain telemetry for Kubernetes clusters and GPU compute farms:

```
import pydataui as pdu

pdu.set_page_config(title="Cluster Telemetry", layout="wide", theme="zinc", dark_mode=True)

with pdu.flex(direction="row", justify="between", align="center", gap=4):
    pdu.subheader("Production Cluster Alpha")
    with pdu.row(align="center", gap=2):
        pdu.badge("US-EAST-1", color="zinc", variant="outline")
        pdu.badge("32/32 PODS HEALTHY", color="emerald", variant="subtle")
        pdu.button("+ Deploy Pod", type="primary")

with pdu.grid(min_column_width="260px", gap=4):
    pdu.stat_card("Active Pods", "32 / 32", change="100% OK", sparkline=[10, 10, 10, 10], color="emerald")
    pdu.stat_card("P99 Latency", "8.2 ms", change="-2.1ms", sparkline=[14, 12, 10, 8], color="indigo")
    pdu.stat_card("VRAM Allocated", "1,180 GB", change="+12%", sparkline=[60, 75, 85, 92], color="amber")
    pdu.stat_card("Token Throughput", "18.4k t/s", change="+24.8%", sparkline=[12, 15, 17, 18], color="rose")

with pdu.card(position="relative"):
    pdu.badge("HOTSPOT DETECTED", color="rose", position="absolute", top="14px", right="14px")
    pdu.title("Worker Node 04")
    pdu.line_chart([65, 72, 80, 85, 94, 98, 82], color="rose")

with pdu.pinned(position="bottom"):
    with pdu.flex(justify="between", align="center", class_name="max-w-4xl mx-auto py-2"):
        pdu.text("3 worker nodes require kernel updates.")
        pdu.button("Apply Batch Update", type="primary")

if __name__ == "__main__":
    pdu.run(port=8000)
```

Recipe 4: Hardened Production FastAPI Microservice with Background Tasks

Full FastAPI Starlette production deployment with CORS, background workers, and Swagger filtering:

```
import pydataui as pdu
from fastapi import APIRouter, BackgroundTasks
from starlette.middleware.cors import CORSMiddleware

# Configure production Swagger UI security
pdu.set_page_config(
    title="Production Hardened Service",
    swagger_ui_parameters={"filter": True, "syntaxHighlight.theme": "monokai"},
    docs_roles=["admin"]
)

# Access native FastAPI application
fastapi_app = pdu.get_fastapi()

# Attach CORS Middleware
fastapi_app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://mycompany.com"],
    allow_credentials=True,
    allow_methods=["GET", "POST"],
    allow_headers=["*"]
)

# Register background task router
router = APIRouter(prefix="/v1")

def audit_worker(action: str):
    print(f"[AUDIT] Logging background action: {action}")

@router.post("/scan")
def trigger_scan(bg: BackgroundTasks):
    bg.add_task(audit_worker, "Initiated full vulnerability scan")
    return {"status": "enqueued", "queue": "security"}

fastapi_app.include_router(router)

pdu.title("Hardened Operations Center")
pdu.write("All endpoints protected with authenticated Swagger UI and CORS.")

if __name__ == "__main__":
    pdu.run(port=8000)
```

Chapter 13: Architectural Trade-Offs & The 5 Limitations

Choosing the right web architecture for a commercial project requires complete clarity about what each tool was engineered to do. Below is a rigorous analysis of **where PyDataUI wins**, its **5 concrete architectural limitations**, and a comprehensive 4-way comparison matrix against Next.js, FastAPI, and Streamlit.

13.1 The 5 Concrete Architectural Limitations vs. Next.js

LIMITATION 1: 1. Zero Offline-First Apps (Network & Server Dependency): Next.js can bundle client-side React code running entirely inside the browser without contacting a server (e.g., offline PWAs, local SQLite via WASM). PyDataUI is *server-driven* via HTMX DOM morphing: interactive callbacks require a connection to the Python ASGI backend. If connectivity drops, dynamic server callbacks pause until reconnected.
Verdict: Not suited for offline-first progressive web apps.

LIMITATION 2: 2. Canvas-Heavy, 120 FPS Client-Side Tools (Figma, Canva, Video Editors): Next.js excels at client-only apps executing thousands of real-time GPU-driven vector calculations running at 120 FPS directly on the client canvas. PyDataUI is engineered for *data products, SaaS platforms, AI copilots, executive dashboards, and B2B portals*; it is not designed to build an in-browser Photoshop or a 3D gaming engine.
Verdict: For browser graphic engines or games, use Next.js / WebGL.

LIMITATION 3: 3. The Vast 3rd-Party NPM React Component Ecosystem: The JavaScript ecosystem has 2M+ NPM packages. If you need a hyper-niche widget (e.g., *react-flow* for node-based graphs, *tiptap* for collaborative rich-text Google Docs), Next.js allows you to install it in seconds. PyDataUI ships with 40+ curated Shadcn/Tailwind enterprise components; niche React widgets require custom web component or iframe encapsulation.
Verdict: If your core differentiator is an existing proprietary React widget library, Next.js has the ecosystem edge.

LIMITATION 4: 4. Python GIL & CPU-Bound Blocking Concurrency: In Node.js or multi-threaded servers, event loops handle concurrency differently. In Python, heavy CPU-bound computation (e.g. multi-minute ML training or massive image transformations) on the main thread will block the async event loop unless explicitly offloaded to *BackgroundTasks*, *multiprocessing*, or Celery/Redis workers.
Verdict: CPU-heavy tasks must use asynchronous background workers.

LIMITATION 5: 5. 0ms Optimistic Client UI Updates: React allows client-side 'optimistic updates' where a button or toggle flips in 0ms on the client before the server acknowledges the write. PyDataUI waits for the server roundtrip (~10ms–25ms over broadband) before morphing the DOM fragment.
Verdict: Highly responsive for 99% of business dashboards and SaaS, but not 0ms client-side optimistic simulation.

13.2 The 5 Architectural Superpowers of PyDataUI

Superpower	How PyDataUI Outperforms Next.js + FastAPI & Streamlit
0.02ms In-Memory Model Access	Your UI and your AI models (PyTorch, LangChain, Ollama, Pandas, Polars) share the exact same Python process memory. Zero JSON serialization overhead, zero internal REST/gRPC latency hops.
1 Language & 1 Runtime	100% pure Python. Zero package.json, zero Webpack/Turbopack bundling, zero npm security vulnerabilities, and 0 MB node_modules (saving ~450 MB).
Surgical HTMX DOM Morphing	Unlike Streamlit (which re-executes lines 1-100 on every user click), PyDataUI dispatches only the targeted state action and surgical DOM swap, eliminating session sync bugs and lag.
Native FastAPI & Auto OpenAPI 3.1	Built directly on FastAPI and Starlette. Every state action can automatically expose an OpenAPI 3.1 REST route with interactive Swagger UI at /docs.
Unified Full-Stack Deployment	1 process, 1 port. Deploy via standard Docker, Uvicorn, or launch an instant secure public HTTPS tunnel in 3.5 seconds with share=True.

13.3 4-Way Comprehensive Architecture Comparison Matrix

Dimension	Next.js	FastAPI	Streamlit	PyDataUI v1.5.0
Languages Required	2 (TypeScript + Python)	1 (Python backend)	1 (Python)	1 (Pure Python Full-Stack)
UI Presentation	React / Virtual DOM	None (API only)	Python widgets	Server-Side HTMX DOM Morph
Execution on Click	Client re-render	N/A (REST API)	Reruns lines 1-100	Surgical swap (0 rerun)
Model / Data Latency	15ms-60ms (JSON hop)	0.02ms (In-Memory)	0.02ms (In-Memory)	0.02ms (Direct RAM pointer)
Build & Dependencies	450 MB+ node_modules	Clean Python	Clean Python	Clean Python (0 MB npm)
REST API & Docs	Manual route handlers	OpenAPI 3.1 Swagger	None	Auto OpenAPI 3.1 Swagger UI
Responsive Layout	Tailwind CSS	None	Primitive columns	Native pdu.flex & pdu.grid
Positioning Engine	Tailwind CSS	None	None	Native props (top, right, etc.)
Public Demos	Vercel / Cloud setup	Cloud hosting	Cloud login required	share=True (3.5s tunnel)
Memory per Session	Low (Client-side)	Stateless	High (Duplicated DFs)	Ultra-Low (Shared memory)

Appendix: Complete API Reference & Cheatsheet

Alphabetical quick reference for all major functions in `pydataui` (`pdu`):

Function Signature	Description & Usage
`pdu.audio(data, format, caption)`	HTML5 native audio player from bytes or URL.
`pdu.badge(label, color, variant)`	Renders tag badge ('subtle', 'solid', 'outline') with preset or hex color.
`pdu.breadcrumb(items, separator)`	Hierarchical navigation breadcrumbs (['Home', 'Projects', 'App']).
`pdu.button(label, on_click, type)`	Button trigger ('primary', 'secondary', 'destructive', 'outline').
`pdu.card(title, subtitle, position)`	Elevated container card supporting relative/absolute positioning.
`pdu.chat_input(placeholder)`	Bottom-docked multiline input with Shift+Enter support.
`pdu.chat_message(role)`	Renders chat bubble ('user' or 'assistant').
`pdu.columns(spec, gap)`	Multi-column layout ([1, 2, 1] or integer count).
`pdu.create_user(u, p, email, roles)`	Registers new user with hashed credentials.
`pdu.data_editor(data, num_rows)`	Interactive in-place editable spreadsheet table with cell blur sync.
`pdu.date_input(label, value, key)`	HTML5 native calendar date picker bound to session_state.
`pdu.date_range_picker(label, val)`	Dual-date selector (start → end) bound to session state tuple.
`pdu.dialog(title, trigger_label)`	Accessible modal dialog overlay.
`pdu.divider(label=None)`	Horizontal divider line with optional centered text label.
`pdu.download_button(label, data, ...)`	Client-side file downloader.
`pdu.expander(label, expanded)`	Collapsible disclosure accordion panel.
`pdu.flex(direction, justify, align, gap)`	Granular CSS Flexbox container context manager.
`pdu.form(key) / pdu.form_submit_button`	Batched input form container preventing intermediate state updates.
`pdu.get_fastapi()`	Returns direct access to the underlying fastapi.FastAPI app instance.
`pdu.grid(columns, min_column_width)`	Responsive CSS Grid container with auto-fit wrapping.
`pdu.login(u, p) / pdu.logout()`	Session cookie authentication manager.
`pdu.metric(label, value, delta)`	Executive KPI metric card with delta indicator.
`pdu.navbar(title, logo, links, ...)`	Commercial application header bar with branding, search, and user menu.
`pdu.page(path, roles, public)`	Multi-page route decorator with built-in RBAC guard.
`pdu.pinned(position='bottom')`	Viewport-docked sticky container.
`pdu.popover(label, position, ...)`	Floating popover card container (with pdu.popover('Filters'):).
`pdu.pricing_card(tier, price, ...)`	SaaS subscription pricing tier card.
`pdu.run(port=8000, share=True)`	Starts ASGI server with optional instant 3.5s public HTTPS tunnel.
`pdu.search_input(placeholder, on_search)`	Real-time debounced search bar with instant HTMX callback.
`pdu.session_state`	Dict-like procedural session state manager.
`pdu.set_page_config(...)`	Configures title, layout, theme, dark mode, docs_url, docs_roles.
`pdu.spacer(height=16)`	Vertical whitespace spacing block.
`pdu.sparkline(data, color, fill)`	Pure inline SVG sparkline graph rendering in ~0.001ms.

`pdu.stat_card(title, value, sparkline)`	Commercial executive stat card with integrated inline SVG sparkline.
`pdu.switch_page(path)`	Programmatic page redirection.
`pdu.tabs(tabs, expand, hide_tab_buttons)`	Tabs container with width distribution and headless wizard mode.
`pdu.theme_toggle()`	Interactive dark/light mode toggle switcher.
`pdu.time_input(label, value, key)`	HTML5 native time picker bound to session_state.
`pdu.tooltip(text, position)`	Accessible hover tooltip popup over children or inline marker.
`pdu.video(data, format, caption)`	HTML5 native video player from bytes or URL.
`pdu.write(obj)`	Polymorphic auto-renderer for text, markdown, dicts, and dataframes.
`pdu.write_stream(stream_generator)`	Real-time LLM token streaming into UI via Server-Sent Events (SSE).

Congratulations! You are now fully equipped to build production-grade web applications, executive commercial dashboards, and AI copilot workbenches using PyDataUI.

■ PYDATAUI ENTERPRISE FRAMEWORK v1.6.2

PyDataUI Developer Cookbook & Reference

The Definitive Full-Stack Python Guide for AI Engineers, Data Scientists & Enterprise Backend Teams

Framework Version: v1.6.2 (Latest Production Release on PyPI)

Core Engine: Native FastAPI + Starlette + FastHTML + HTMX DOM Morphing + SSE Real-Time Streaming

v1.6.2 Additions: Scoped UI Fragments (@pdu.fragment), Spatial Hero Composition, Animated Loader Presets, 3D WebGL / AR, Guided Wizards, Zero-Latency Client Directives, ContextVar Thread Safety

Package Footprint: Pure Python wheel (~310 KB) • Zero npm dependencies • 0 MB node_modules

Official PyPI Link: https://pypi.org/project/pydataui/1.6.2/

Table of Contents

1. Architecture & 10-Second Quickstart	FastAPI Native Engine, Zero-Class Mental Model, 10-second runnable cluster dashboard
2. The Dual-Track Engine	Declarative Spatial Track (Next.js-like) vs. Procedural Track (Streamlit-like DX)
3. Advanced Spatial Composition & Hero	with pdu.hero(): multi-layer z-indexes, full-bleed canvases, gradients, and typography
4. Scoped UI Fragments (@pdu.fragment)	Sub-second partial DOM updates with run_every interval and zero top-to-bottom re-runs
5. Guided Multi-Step Wizards & Client Directives	pdu.wizard(steps=[...]), client tab switching, and zero-latency client state directives
6. Animated Loader Presets & Visual Polish	pdu.loader(preset='dots'|'circle'|'pulse'|'bars'|'shimmer'), inline SVG sparklines
7. Interactive Inputs, Data Editor & File Uploader	In-place editable spreadsheets, lazy dataframe proxy uploader with onchange callbacks
8. 3D WebGL, AR & Spatial Data Visualization	pdu.model_viewer(src, auto_rotate=True, ar=True), responsive KPI grid, delta metrics
9. Enterprise AI Copilot & Real-Time Streaming	pdu.chat_message, pdu.chat_input, and real-time LLM token streaming via pdu.write_stream
10. Enterprise FastAPI Mounting & Security	Subpath mounting (app.mount('/ops', ...)), CSRF token protection, RBAC, ContextVar isolation
11. Production Theming Engine (20 Presets)	Zinc, Emerald, Violet, Slate, high-contrast dark mode, and custom HEX brand palettes
12. Production Cookbook: 5 Complete Applications	AI Telemetry Copilot, Scoped Polling Dashboard, Spatial 3D Hero, FinTech Ledger, Mounted FastAPI
13. Architectural Trade-Offs & 4-Way Comparison	Rigorous side-by-side analysis: Next.js vs FastAPI vs Streamlit vs PyDataUI v1.6.2
Appendix: Complete API Reference (v1.6.2)	Exhaustive parameter breakdown for all pdu functions, decorators, and context managers

Chapter 1: Architecture, Mental Model & 10-Second Quickstart

For years, Python teams faced an agonizing trade-off: use **Streamlit** for rapid prototypes (and endure sluggish top-to-bottom script re-runs, fragile session state bugs, and non-existent production REST APIs), or switch to **Next.js + FastAPI** (gaining performance and production APIs at the cost of 450 MB of node_modules, dual-language TypeScript/Python overhead, and endless schema glue code).

PyDataUI v1.6.2 eliminates this dilemma completely. On the surface, you write procedural, top-to-bottom Python with zero boilerplate. Underneath, a production ASGI engine powered by **FastAPI, Starlette, FastHTML, and HTMX DOM morphing** executes surgically. When a user interacts with a widget, only the affected DOM fragment mutates in place—the Python process never re-executes lines 1 to 100 from scratch.

KEY ARCHITECTURAL ADVANTAGE: 0.02ms In-Memory Model Access: Because your UI and your PyTorch, Hugging Face, LangChain, or Pandas models share the exact same Python process memory, you never incur JSON serialization hops or HTTP networking overhead. DataFrames and GPU weights remain in RAM.

10-Second Quickstart Application (`app.py`)

Install via `pip install pydataui` and run with `python app.py`:

```
import pydataui as pdu
import pandas as pd

# 1. Page Configuration & Theme
pdu.set_page_config(title="AI & Sales Intelligence", layout="wide", theme="zinc", dark_mode=True)

# 2. Modern Sidebar Controls
with pdu.sidebar():
    pdu.title("Controls")
    pdu.theme_toggle()
    model = pdu.selectbox("Select Model", ["Llama-3.3", "DeepSeek-R1", "GPT-4o"])
    temp = pdu.slider("Temperature", 0.0, 1.0, 0.70)
    dataset = pdu.file_uploader("Upload Evaluation CSV", accept=[".csv"])
    pdu.write(dataset.get_data())

# 3. Main Dashboard Header & Status
pdu.title("GPU Cluster Telemetry & Agent Dispatch")
pdu.success("All 16 H100 SXM5 nodes online. Telemetry streaming active.")

# 4. Multi-Column KPI Grid
col1, col2, col3 = pdu.columns([1, 1, 1])
with col1:
    pdu.metric("Active Sessions", "2,840", delta="+12.4%")
with col2:
    pdu.metric("P99 Inference Latency", "14.2ms", delta="-3.1ms")
with col3:
    pdu.metric("Token Throughput", "8.2k tokens/s", delta="+18.5%")

# 5. Interactive Tabs & Real-Time Copilot
tab_data, tab_chat = pdu.tabs(["Data Explorer", "Operations Copilot"])
with tab_data:
    df = pd.DataFrame({
        "Worker Node": [f"node-{i:02d}" for i in range(1, 5)],
        "GPU Utilization": ["88%", "92%", "78%", "95%"],
        "VRAM Allocated": ["72 GB", "76 GB", "64 GB", "78 GB"]
    })
    pdu.write(df)  # Polymorphic auto-renderer: interactive paginated table
    pdu.line_chart([65, 72, 80, 85, 92, 88, 91, 95])

with tab_chat:
    pdu.title("Chat Interface")
    with pdu.chat_message("user"):
        pdu.write("Are there any thermal anomalies on worker nodes?")
    with pdu.chat_message("assistant"):
        pdu.write("All nodes operating within normal thermal limits. Peak VRAM was 78 GB on Node-04.")
    pdu.chat_input("Ask the operations assistant a question...")

if __name__ == "__main__":
    pdu.run(port=8000)  # Ready in 3.5s with instant public HTTPS tunnel!
```

Chapter 2: The Dual-Track Engine: Next.js vs. Streamlit Paradigm

PyDataUI provides a **Dual-Track Architecture** that allows developers to choose between two fundamentally distinct mental models depending on project requirements:

Dimension	Track 1: Procedural DX (Streamlit Track)	Track 2: Declarative Track (Next.js Track)
Mental Model	Top-to-bottom procedural script flow (`pdu.write`, `pdu.title`).	Hierarchical spatial component composition tree.
Syntax Style	Flat, sequential calls with context managers (`with pdu.sidebar():`).	Composable functions, props, and explicit children slots.
Layout Flow	1D vertical document stack with automatic column grouping.	Full 2D spatial coordinate positioning (z-index, fixed overlays).
State Handling	Transparent session dictionary (`pdu.session_state['key']`).	Explicit immutable state flow with reactive action dispatch.
Target Audience	Data scientists, AI researchers, internal tool builders.	Full-stack engineers, SaaS builders, commercial app designers.
Re-execution Scope	Scoped HTMX DOM morph (zero full-script re-run).	Granular component-level virtual DOM diffing and updates.

Understanding The Fundamental Paradox

In traditional Streamlit, you cannot stack elements along the Z-axis (e.g., placing text over a looping background video or layering a glassmorphic badge over an interactive map). In Next.js, doing this requires configuring Tailwind z-indexes, absolute position wrappers, and multiple JSX divs.

PyDataUI bridges both worlds: you get the spatial composition capabilities of Next.js through high-level Python abstractions like **`pdu.hero()`**, **`pdu.pinned()`**, and coordinate positioning props (`position='absolute'`, `top='12px'`), while keeping 100% pure Python procedural syntax.

Chapter 3: Advanced Spatial Composition & Multi-Layer Hero

In v1.6.2, PyDataUI introduces first-class spatial composition containers. Rather than being confined to sequential stacked boxes, you can compose multi-layered spatial experiences with full-bleed backgrounds, dark gradient overlays, and typography layers in seconds.

Multi-Layer Hero Section (`with pdu.hero():`)

```
import pydataui as pdu

pdu.set_page_config(title="SaaS Platform", layout="wide", theme="violet", dark_mode=True)

# 1. Full-Bleed Multi-Layer Hero Canvas
with pdu.hero(
    bg_video="https://assets.mixkit.co/videos/preview/mixkit-digital-animation-of-screens-4192-large.mp4",
    overlay_opacity=0.45,
    min_height="520px",
    align="center",
    justify="center"
):
    pdu.badge("RELEASED v1.6.2", color="violet", variant="subtle")
    pdu.title("Autonomous AI Infrastructure for Enterprise Teams")
    pdu.subheader("Deploy, monitor, and scale self-healing agent clusters in pure Python.")

    with pdu.row(gap=4, justify="center"):
        pdu.button("Start Free Trial", type="primary")
        pdu.button("Read Architecture Whitepaper", type="outline")

# 2. Docked Coordinate Badges & Sticky Viewport Toolbars
with pdu.card(position="relative"):
    pdu.badge("LIVE TELEMETRY", color="emerald", position="absolute", top="14px", right="14px")
    pdu.subheader("Distributed Inference Pods")
    pdu.line_chart([78, 82, 85, 91, 88, 94, 96])

with pdu.pinned(position="bottom"):
    with pdu.flex(justify="between", align="center", class_name="max-w-5xl mx-auto py-2"):
        pdu.write("Cluster health: **100% Operational** • 0 dropped packets")
        pdu.button("Drain Selected Nodes", type="destructive")
```

Layout Primitives: `pdu.flex` and `pdu.grid`

Unlike primitive fixed-width columns, PyDataUI provides production CSS Flexbox and Grid wrappers with responsive wrapping, alignment controls, and gap tokens:

```
# Flexible row layout with space-between distribution
with pdu.flex(direction="row", justify="between", align="center", gap=4):
    pdu.title("Operations Hub")
    pdu.date_input("Filter Date", value="2026-11-19")

# Responsive auto-wrapping grid (min column width 280px)
with pdu.grid(min_column_width="280px", gap=4):
    pdu.metric("API Requests", "1.24M", delta="+8.4%")
    pdu.metric("Cache Hit Ratio", "94.2%", delta="+1.1%")
    pdu.metric("Active Workers", "64 / 64", delta="Optimal")
```

Chapter 4: Scoped UI Fragments (@pdu.fragment)

One of the biggest breakthroughs in PyDataUI v1.6.2 is **Scoped UI Fragments**. In standard frameworks, any background polling or real-time ticker triggers a full script execution or page repaint, consuming high CPU and resetting scroll position. With `@pdu.fragment`, only the wrapped function executes and refreshes its dedicated DOM container.

```
import pydataui as pdu
import random
import time

pdu.set_page_config(title="Real-Time Fragment Polling", theme="zinc", dark_mode=True)
pdu.title("Real-Time Cluster Telemetry")

# Static control panel rendered once
with pdu.sidebar():
    pdu.selectbox("Telemetry Metric", ["VRAM Usage", "Core Temperature", "Power Draw"])
    pdu.button("Clear Buffer")

# Scoped Fragment: Only this block re-runs every 2.0 seconds!
@pdu.fragment(run_every=2.0)
def live_gpu_metrics():
    current_temp = random.randint(68, 84)
    vram_used = random.uniform(64.2, 79.8)
    delta_val = f"{random.uniform(-1.5, 2.5):+.1f}°C"

    col1, col2 = pdu.columns([1, 1])
    with col1:
        pdu.metric("H100 Core Temp", f"{current_temp}°C", delta=delta_val)
    with col2:
        pdu.metric("VRAM Consumption", f"{vram_used:.1f} GB", delta="+0.4 GB")

    pdu.line_chart([random.randint(60, 95) for _ in range(10)], color="violet")
    pdu.caption(f"Last polled at: {time.strftime('%H:%M:%S')}")

# Invoke fragment container
live_gpu_metrics()

pdu.write("This footer text is completely static and never flickers during fragment polls.")
```

FRAGMENT EFFICIENCY: Zero Full-Script Re-runs: When `live_gpu_metrics()` triggers via SSE or HTMX timer, the Python process dispatches only that fragment callback. Static controls, inputs, and other cards on the page remain completely untouched.

Chapter 5: Guided Multi-Step Wizards & Client Directives

Complex enterprise workflows—such as multi-stage tenant onboarding, checkout funnels, and model evaluation pipelines—require structured step navigation. PyDataUI provides **`pdu.wizard`** and **Zero-Latency Client Directives** for instant UI interactions without network round-trips.

```
import pydataui as pdu

pdu.set_page_config(title="Onboarding Wizard", theme="emerald")

# Guided Multi-Step Wizard with Step Navigation
wizard = pdu.wizard(steps=["Cluster Details", "GPU Sizing", "Security & Keys", "Review"])

if wizard.current_step == 0:
    pdu.subheader("Step 1: Cluster Configuration")
    cluster_name = pdu.text_input("Cluster Name", value="alpha-prod-cluster")
    region = pdu.selectbox("Deployment Region", ["us-east-1", "eu-central-1", "ap-southeast-1"])
    if pdu.button("Next: GPU Sizing →", type="primary"):
        wizard.next_step()

elif wizard.current_step == 1:
    pdu.subheader("Step 2: Compute & Accelerator Selection")
    gpu_type = pdu.selectbox("GPU Accelerator", ["NVIDIA H100 SXM5", "A100 80GB", "L40S"])
    node_count = pdu.slider("Number of Worker Nodes", 1, 32, 4)
    with pdu.row(gap=2):
        if pdu.button("← Previous", type="outline"):
            wizard.previous_step()
        if pdu.button("Next: Security →", type="primary"):
            wizard.next_step()

elif wizard.current_step == 2:
    pdu.subheader("Step 3: Security & Access Control")
    pdu.checkbox("Enable Mutual TLS (mTLS)", value=True)
    pdu.checkbox("Enforce RBAC Strict Mode", value=True)
    with pdu.row(gap=2):
        if pdu.button("← Previous", type="outline"):
            wizard.previous_step()
        if pdu.button("Next: Review →", type="primary"):
            wizard.next_step()

elif wizard.current_step == 3:
    pdu.success("Configuration validated! Ready for provision.")
    pdu.button("■ Launch Production Cluster", type="primary")
```

Zero-Latency Client Directives (`pdu.client_toggle`)

For instant visual interactions (e.g. expanding collapsible drawers or toggling client-side view filters), client directives run directly in the browser DOM in 0.0ms without waiting for a server network round-trip:

```
# Toggle visibility of a target element with zero network latency
pdu.client_toggle(trigger_label="Toggle Advanced Debug Trace", target_id="debug-panel")
with pdu.card(id="debug-panel", class_name="hidden"):
    pdu.write("Raw CUDA kernel assembly & memory allocation map...")
```

Chapter 6: Animated Loader Presets & Visual Polish

User experience during asynchronous processing (such as LLM inference, SQL queries, or file uploads) relies heavily on visual feedback. In v1.6.2, PyDataUI includes **5 built-in Animated Loader Presets** matching modern SaaS design standards.

Preset Name	Visual Animation Pattern	Ideal Production Use Case
dots	Three staggered bouncing dots with easing curve	Chatbot thinking indicator, quick LLM token prep
circle	Rotating SVG spinner with tapered gradient trail	Page-level initialization, auth verification
pulse	Concentric rhythmic pulsing glow	Live streaming heartbeat, sensor telemetry wait
bars	Five ascending and descending equalizer bars	Audio processing, transcription, vector ingestion
shimmer	Continuous horizontal skeleton shimmer placeholder	Data table pre-fetching, card loading skeleton

Loader Usage Examples

```
import pydataui as pdu
import time

# 1. Staggered Bouncing Dots
pdu.loader(preset="dots", text="Synthesizing response from DeepSeek-R1...")

# 2. Concentric Pulse Indicator
pdu.loader(preset="pulse", text="Awaiting cluster node heartbeat...", color="emerald")

# 3. Audio / Vector Equalizer Bars
pdu.loader(preset="bars", text="Embedding 45,000 vector chunks into ChromaDB...", color="violet")

# 4. Context Manager Wrapper
with pdu.spinner("Executing Monte Carlo portfolio simulation..."):
    time.sleep(1.5)
    pdu.success("Simulation completed across 10,000 runs.")
```

Inline SVG Sparklines (0.001ms Render Time)

PyDataUI includes a pure Python inline SVG sparkline generator with zero external JavaScript charting dependencies. It renders in under 1 microsecond:

```
# Render standalone inline SVG sparkline
pdu.sparkline([45, 52, 58, 64, 72, 69, 81, 94], color="#10B981", fill=True, height=36)

# Render inside an executive Stat Card
pdu.stat_card(
    title="ARR Run Rate",
    value="$1,840,500",
    change="+24.8% YoY",
    sparkline=[120, 135, 148, 162, 184],
    color="emerald"
)
```

Chapter 7: Interactive Inputs, Data Editor & File Uploader

PyDataUI v1.6.2 includes high-density data controls engineered specifically for data science and machine learning workflows, including **in-place editable spreadsheets** and an enhanced **File Uploader with Lazy Data Proxy**.

Interactive Data Editor (`pdu.data_editor`)

Renders a spreadsheet table where users can edit cell values directly in the browser with automatic blur synchronization back to Python:

```
import pydataui as pdu
import pandas as pd

df = pd.DataFrame({
    "Hyperparameter": ["learning_rate", "batch_size", "warmup_steps", "weight_decay"],
    "Value": ["0.0002", "64", "500", "0.01"],
    "Optimizable": [True, False, True, True]
})

# In-place editable spreadsheet table
edited_df = pdu.data_editor(df, num_rows="dynamic", key="hp_table")

if pdu.button("Apply Hyperparameters", type="primary"):
    pdu.write(f"Updated configuration applied: {edited_df}")
```

Enhanced File Uploader with Lazy Data Proxy

In v1.6.2, `pdu.file_uploader` returns a lazy data proxy. Calling `dataset.get_data()` safely renders an empty state before upload and seamlessly resolves into a full Pandas DataFrame once uploaded:

```
# File Uploader with instant onchange callback and lazy proxy
dataset = pdu.file_uploader(
    "Upload Evaluation CSV",
    accept=[".csv", ".parquet"],
    onchange=lambda file: pdu.toast(f"Uploaded {file.name}")
)

# Polymorphic auto-renderer safely handles empty proxy before upload
# and displays the full interactive table once the file is uploaded:
pdu.write(dataset.get_data())
```

Chapter 8: 3D WebGL, AR & Spatial Data Visualization

Enterprise physical AI, robotics, and hardware telemetry require interactive 3D rendering. PyDataUI v1.6.2 integrates browser-native WebGL and Augmented Reality (AR) without needing Three.js boilerplate.

Browser-Native 3D Model Viewer (`pdu.model_viewer`)

```
import pydataui as pdu

pdu.set_page_config(title="Robotics & Physical AI Telemetry", layout="wide", theme="zinc", dark_mode=True)

col_3d, col_telemetry = pdu.columns([3, 2])

with col_3d:
    pdu.subheader("Robotic Arm Digital Twin (SDF / glTF)")
    # Native WebGL 3D model viewer with auto-rotation, orbit controls, and AR pass-through
    pdu.model_viewer(
        src="https://modelviewer.dev/shared-assets/models/Astronaut.glb",
        poster="https://modelviewer.dev/shared-assets/models/Astronaut.webp",
        auto_rotate=True,
        camera_controls=True,
        ar=True,
        height="480px"
    )

with col_telemetry:
    pdu.subheader("Actuator Sensor Array")
    pdu.metric("Joint 1 Torque", "42.8 Nm", delta="+1.2 Nm")
    pdu.metric("Joint 2 Velocity", "1.4 rad/s", delta="Nominal")
    pdu.metric("End-Effector Temp", "38.2°C", delta="-0.8°C")

    pdu.spacer(height=12)
    pdu.button("Recalibrate Actuators", type="primary")
```

Chapter 9: Enterprise AI Copilot & Real-Time Token Streaming

Modern enterprise applications require conversational copilot interfaces. PyDataUI includes built-in chat bubbles, docking multiline inputs, and real-time Server-Sent Events (SSE) token streaming via `pdu.write_stream`.

```
import pydataui as pdu
import time

pdu.set_page_config(title="AI Copilot Workbench", theme="violet", dark_mode=True)
pdu.title("Autonomous Cloud Copilot")

# Chat History Container
with pdu.chat_message("user"):
    pdu.write("Why did node-03 experience a latency spike at 14:22 UTC?")

with pdu.chat_message("assistant"):
    with pdu.expander("■ Reasoning Trace: 2,100 tokens analyzed"):
        pdu.write("Inspected Prometheus logs. Kernel page cache compaction locked worker threads.")

    # Real-time token generator
    def token_stream():
        response = "Node-03 experienced memory compaction under heavy batch evaluation. Remediated by reallocating pods to node-0
        for word in response.split(" "):
            yield word + " "
            time.sleep(0.04)

    # Streams tokens dynamically into the browser without full reloads
    pdu.write_stream(token_stream)

# Docked multiline chat input with Shift+Enter support
pdu.chat_input("Ask Copilot or type /diagnose...")
```

Chapter 10: Enterprise FastAPI Mounting & Production Security

Unlike standalone dashboard tools that force you to run separate servers for APIs and UI, PyDataUI is **FastAPI-native**. You can mount PyDataUI as a sub-application inside an existing enterprise FastAPI or Starlette router with full CSRF protection and Role-Based Access Control (RBAC).

```
import pydataui as pdu
from fastapi import FastAPI

# 1. Existing Enterprise FastAPI Application
main_app = FastAPI(title="Enterprise Gateway")

@main_app.get("/api/health")
def health_check():
    return {"status": "healthy", "service": "gateway"}

# 2. Define PyDataUI Dashboard Route with RBAC Protection
@pdu.page("/ops", roles=["admin", "sre"])
def cluster_ops_page():
    pdu.title("Protected SRE Command Center")
    pdu.metric("Cluster Availability", "99.999%", delta="Stable")
    pdu.write("Restricted to users with 'admin' or 'sre' roles.")

# 3. Mount PyDataUI directly into existing FastAPI app
# In v1.6.2, static assets automatically resolve to /ops/_pdu/static/
main_app.mount("/ops", pdu.get_fastapi())

# Run with: uvicorn app:main_app --host 0.0.0.0 --port 8000
```

Security Architecture Highlights (v1.6.2)

Security Vector	Implementation in PyDataUI v1.6.2
CSRF Defense	Automatic CSRF tokens injected into all form actions, HTMX requests, and file uploads.
Asset Prefixing	Static assets dynamically prefix with root_path (e.g. `/ops/_pdu/static/`) when mounted.
ContextVar Isolation	Strict `try...finally` session cleanup guarantees zero cross-test or cross-request state leakage.
Cookie Security	HttpOnly, SameSite=Lax, and Secure session cookies prevent XSS credential exfiltration.
Role-Based Access (RBAC)	Route-level guards (`@pdu.page(roles=['admin'])`) reject unauthorized sessions with 403 Forbidden.

Chapter 11: Production Theming Engine (20 Presets & Custom HEX)

PyDataUI ships with **20 official color presets** based on modern design systems (Shadcn UI & Tailwind CSS tokens). Every theme supports light mode, high-contrast dark mode, and custom HEX brand palettes in 1 line of Python.

Preset Name	Category	Hex Primary	Ideal Application Domain
zinc	Modern Neutrals	#18181B	DevOps, Kubernetes Clusters, GPU Telemetry
violet	Blues & Purples	#7C3AED	AI Agent Copilots, LLM Workbenches, Chatbots
emerald	Greens & Cyans	#059669	FinTech, SaaS Billing, Revenue Analytics
indigo	Blues & Purples	#4F46E5	Default Enterprise SaaS, Admin Portals
amber	Pinks & Warm	#D97706	Logistics, Supply Chain, Warehouse Fleet
rose	Pinks & Warm	#E11D48	Incident Management, Alerting, Security Operations
slate	Modern Neutrals	#0F172A	Executive Dashboards, Business Intelligence
cyan	Greens & Cyans	#0891B2	Cloud Infrastructure, Storage Telemetry
blue	Blues & Purples	#2563EB	Corporate Portals, Developer Documentation
stone	Modern Neutrals	#1C1917	Publishing, Content Management Systems

Theming Code Examples

```
import pydataui as pdu

# 1. Preset with Dark Mode
pdu.set_page_config(theme="zinc", dark_mode=True)

# 2. Custom Brand Hex
pdu.set_page_config(theme="#4f46e5")

# 3. Interactive Theme Switcher Button
pdu.theme_toggle()
```

Chapter 12: Production Cookbook: 5 Complete Applications

Recipe 1: AI Cluster Telemetry & SRE Workbench (`theme='zinc'`)

```
import pydataui as pdu
import pandas as pd

pdu.set_page_config(title="SRE Cluster Hub", layout="wide", theme="zinc", dark_mode=True)

with pdu.sidebar():
    pdu.title("Telemetry Controls")
    pdu.theme_toggle()
    pdu.selectbox("Active Region", ["us-east-1", "eu-central-1"])
    dataset = pdu.file_uploader("Upload Evaluation Log", accept=[".csv"])
    pdu.write(dataset.get_data())

pdu.title("GPU Cluster Telemetry & Node Orchestration")
pdu.success("All 16 H100 SXM5 compute nodes healthy. Ingestion stream active.")

col1, col2, col3 = pdu.columns([1, 1, 1])
with col1:
    pdu.metric("Active Sessions", "2,840", delta="+12.4%")
with col2:
    pdu.metric("P99 Inference Latency", "14.2ms", delta="-3.1ms")
with col3:
    pdu.metric("Token Throughput", "8.2k tokens/s", delta="+18.5%")

tab_metrics, tab_chat = pdu.tabs(["Node Telemetry", "Copilot Assistant"])
with tab_metrics:
    df = pd.DataFrame({
        "Worker Node": [f"node-{i:02d}" for i in range(1, 5)],
        "GPU Utilization": ["88%", "92%", "78%", "95%"],
        "VRAM Allocated": ["72 GB", "76 GB", "64 GB", "78 GB"]
    })
    pdu.write(df)
    pdu.line_chart([65, 72, 80, 85, 92, 88, 91, 95])

with tab_chat:
    with pdu.chat_message("user"):
        pdu.write("Diagnose VRAM spike on worker node-04.")
    with pdu.chat_message("assistant"):
        pdu.write("Node-04 peaked at 78 GB due to KV cache pre-allocation. Stable within headroom.")
    pdu.chat_input("Ask Copilot...")

if __name__ == "__main__":
    pdu.run(port=8000)
```

Recipe 2: Real-Time Sub-Second Polling Fragment (`@pdu.fragment`)

```
import pydataui as pdu
import random
import time

pdu.set_page_config(title="Live SRE Monitor", theme="zinc", dark_mode=True)
pdu.title("High-Frequency Cluster Ticker")

@pdu.fragment(run_every=1.5)
def live_node_ticker():
    p99 = random.uniform(8.1, 14.8)
    delta_str = f"{random.uniform(-1.2, 1.5):+.1f}ms"
    pdu.metric("P99 Cluster Latency", f"{p99:.1f}ms", delta=delta_str)
    pdu.line_chart([random.randint(50, 95) for _ in range(8)], color="violet")
    pdu.caption(f"Last updated: {time.strftime('%H:%M:%S')}")

live_node_ticker()
pdu.write("Static controls on this page never re-run or reload during polling intervals.")

if __name__ == "__main__":
    pdu.run(port=8000)
```

Recipe 3: Spatial Hero Landing Page & 3D WebGL Digital Twin

```
import pydataui as pdu

pdu.set_page_config(title="Hardware AI Twin", layout="wide", theme="violet", dark_mode=True)

with pdu.hero(
    bg_video="https://assets.mixkit.co/videos/preview/mixkit-digital-animation-of-screens-4192-large.mp4",
    overlay_opacity=0.40,
    min_height="480px"
):
    pdu.badge("DIGITAL TWIN v1.6.2", color="violet")
    pdu.title("Physical AI & Robotics Telemetry")
    pdu.subheader("Real-time 3D telemetry streaming directly into browser WebGL.")

col_left, col_right = pdu.columns([3, 2])
with col_left:
    pdu.model_viewer("https://modelviewer.dev/shared-assets/models/Astronaut.glb", auto_rotate=True)
with col_right:
    pdu.stat_card("Sensor Throughput", "48.2 kHz", change="+12%", sparkline=[40, 42, 45, 48])
    pdu.stat_card("Actuator Temp", "41.2°C", change="-0.4°C", sparkline=[44, 43, 42, 41])

if __name__ == "__main__":
    pdu.run(port=8000)
```

Chapter 13: Architectural Trade-Offs & 4-Way Comparison

Selecting the right web framework requires absolute engineering clarity. Below is an objective analysis of PyDataUI v1.6.2 compared directly against Next.js, FastAPI, and Streamlit:

Dimension	Next.js	FastAPI	Streamlit	PyDataUI v1.6.2
Languages Required	2 (TypeScript + Python)	1 (Python backend)	1 (Python)	1 (Pure Python Full-Stack)
UI Presentation	React / Virtual DOM	None (API only)	Python widgets	FastHTML + HTMX DOM Morph
Execution on Click	Client re-render	N/A (REST API)	Reruns lines 1-100	Surgical fragment morph (0 rerun)
Model / Data Latency	15ms-60ms (JSON hop)	0.02ms (In-Memory)	0.02ms (In-Memory)	0.02ms (Direct RAM pointer)
Build & Dependencies	450 MB+ node_modules	Clean Python	Clean Python	Clean Python (0 MB npm)
REST API & Docs	Manual route handlers	OpenAPI 3.1 Swagger	None	Auto OpenAPI 3.1 Swagger UI
Multi-Layer Hero	Tailwind CSS JSX	None	Impossible (1D Flow)	Native `with pdu.hero():`
3D WebGL / AR	Three.js / React Fiber	None	Component wrapper	Native `pdu.model_viewer`
Scoped Polling	React useEffect	N/A	st.empty() loop	`@pdu.fragment(run_every=2)`
Public Demos	Cloud hosting setup	Cloud hosting	Cloud login required	`share=True (3.5s HTTPS link)`
Memory per Session	Low (Client-side)	Stateless	High (Duplicated DFs)	Ultra-Low (Shared memory)

When To Choose Which Tool

Framework	Ideal Choice When...
PyDataUI v1.6.2	You need production-grade web applications, commercial AI dashboards, or SaaS portals in pure Python with 0ms in-memory model access and zero npm/Node.js dependencies.
Next.js	You are building consumer-facing apps requiring 120 FPS canvas tools (Figma, Canva), offline PWAs, or proprietary React widget libraries.
Streamlit	You need a disposable throwaway script for a 1-hour internal exploratory spike where production APIs and custom layouts do not matter.
FastAPI	You are exposing purely headless REST/JSON APIs for third-party microservices without requiring an integrated UI.

Appendix: Complete API Reference & Cheatsheet (v1.6.2)

Alphabetical quick reference for all major functions, context managers, and decorators in `pydataui` (`pdu`):

Function Signature	Description & Usage
`pdu.badge(label, color, variant, position)`	Renders tag badge ('subtle', 'solid', 'outline') with coordinate positioning.
`pdu.button(label, on_click, type)`	Interactive button trigger ('primary', 'secondary', 'destructive', 'outline').
`pdu.card(title, subtitle, position)`	Elevated container card supporting relative/absolute positioning.
`pdu.chat_input(placeholder)`	Bottom-docked multiline input with Shift+Enter support.
`pdu.chat_message(role)`	Renders chat bubble ('user' or 'assistant').
`pdu.client_toggle(trigger_label, target_id)`	Zero-latency client-side DOM toggle without server roundtrip.
`pdu.columns(spec, gap)`	Multi-column layout ([1, 2, 1] or integer count).
`pdu.data_editor(data, num_rows, key)`	Interactive in-place editable spreadsheet table with cell blur sync.
`pdu.date_input(label, value, key)`	HTML5 native calendar date picker bound to session_state.
`pdu.divider(label=None)`	Horizontal divider line with optional centered text label.
`pdu.expander(label, expanded)`	Collapsible disclosure accordion panel.
`pdu.file_uploader(label, accept, onchange)`	File uploader returning lazy data proxy with automatic table rendering.
`pdu.flex(direction, justify, align, gap)`	Granular CSS Flexbox container context manager.
@pdu.fragment(run_every=None)	Scoped UI fragment decorator for partial DOM updates and sub-second polling.
`pdu.get_fastapi()`	Returns direct access to the underlying fastapi.FastAPI app instance.
`pdu.grid(columns, min_column_width, gap)`	Responsive CSS Grid container with auto-fit wrapping.
with pdu.hero(bg_video, overlay_opacity)	Multi-layer spatial hero container with full-bleed canvas and z-index layers.
`pdu.line_chart(data, color, height)`	Interactive responsive line chart.
`pdu.loader(preset, text, color)`	Animated loader ('dots', 'circle', 'pulse', 'bars', 'shimmer').
`pdu.metric(label, value, delta, delta_type)`	Executive KPI metric card with positive/negative color styling.
`pdu.model_viewer(src, auto_rotate, ar)`	Native WebGL 3D model viewer and Augmented Reality container.
`pdu.page(path, roles, public)`	Multi-page route decorator with built-in RBAC guard.
`pdu.pinned(position='bottom')`	Viewport-docked sticky container toolbar.
`pdu.run(port=8000, share=True)`	Starts ASGI server with optional instant 3.5s public HTTPS tunnel.
`pdu.session_state`	Dict-like procedural session state manager with ContextVar safety.
`pdu.set_page_config(...)`	Configures title, layout, theme, dark mode, docs_url, docs_roles.
`pdu.sparkline(data, color, fill)`	Pure inline SVG sparkline graph rendering in ~0.001ms.
`pdu.stat_card(title, value, sparkline)`	Commercial executive stat card with integrated inline SVG sparkline.
`pdu.tabs(tabs, expand, hide_tab_buttons)`	Tabs container with width distribution and headless wizard mode.
`pdu.theme_toggle()`	Interactive dark/light mode toggle switcher.
`pdu.wizard(steps=[...])`	Guided multi-step wizard with next_step() and previous_step().
`pdu.write(obj)`	Polymorphic auto-renderer for text, markdown, dicts, dataframes, and proxies.
`pdu.write_stream(stream_generator)`	Real-time LLM token streaming into UI via Server-Sent Events (SSE).

Congratulations! You are now fully equipped to build production-grade web applications, executive commercial dashboards, and AI copilot workbenches using PyDataUI v1.6.2.