## Blue Prism Job Tracking – Overall Workflow (Structured Overview)

## 1. Objective

To \*\*monitor Blue Prism process execution\*\* (jobs/tasks) \*\*outside\*\* of the Blue Prism Control Room using a \*\*Python-based web dashboard\*\* that displays:

* Number of total, running, completed, failed, and pending tasks
* Last executed and next scheduled jobs
* Real-time progress and detailed stats per process

## 2. High-Level Architecture

The architecture consists of several layers:

**Blue Prism System** (Processes & Scheduler) → Data Extraction Layer (bp\_connector.py) → Data Storage Layer (SQLite - tracker.db) → Application Layer (Flask Backend - app.py) → Presentation Layer (HTML + JS Frontend) → **End User** (Views job status on browser)

The **Data Extraction Layer** fetches job/process data via API or SQL DB.  
The **Data Storage Layer** stores job execution logs & metadata.  
The **Application Layer** exposes REST endpoints, aggregates data, and serves the dashboard.  
The **Presentation Layer** displays the dashboard UI with overall stats, per-task progress, and last/next execution details.

## 3. Workflow Breakdown (Step-by-Step)

### Step 1: Blue Prism Execution

Blue Prism runs multiple processes/jobs (either manually, scheduled, or triggered).  
Each job’s metadata (start time, end time, status, etc.) is stored internally in Blue Prism DB or available via Blue Prism API.

### Step 2: Data Extraction Layer (Python Connector)

A \*\*Python module\*\* (`bp\_connector.py`) retrieves job-level details from Blue Prism through one of two channels:

* **API Method** → Using Blue Prism REST API endpoints (e.g., `/sessions`, `/processes`).
* **Database Method** → Reading directly from the \*\*Blue Prism SQL Database\*\* tables (e.g., `BPASession`, `BPAProcess`).

Data captured includes:

* Process Name
* Session ID
* Status (Running, Completed, Failed, Pending)
* Start Time / End Time
* Next Scheduled Execution Time
* Optional: progress percentage or metrics

### Step 3: Data Storage Layer (SQLite)

Data fetched from Blue Prism is stored locally in \*\*SQLite database\*\* (`tracker.db`).  
Each job’s execution details are logged as a historical record for audit or trend visualization.  
This layer ensures:

* **Persistence** (data available even after restart)
* **Historical analysis**
* **Faster retrieval** for the UI

### Step 4: Application Layer (Flask Backend)

A \*\*Flask web server\*\* acts as the central middleware and API layer.  
Its responsibilities:

* Fetch the latest job data from Blue Prism (through connector)
* Store/Update job logs into SQLite
* Compute key \*\*aggregate metrics\*\*, such as:
  + Total number of tasks
  + Count of Completed / Running / Failed / Pending
  + Last executed process name
  + Next scheduled process name
* Expose \*\*RESTful APIs\*\* (e.g., `/api/status`, `/api/logs`) for front-end consumption
* Serve the \*\*web dashboard (HTML template)\*\* to the user

### Step 5: Presentation Layer (Web Dashboard UI)

A \*\*lightweight HTML + JavaScript\*\* dashboard dynamically calls Flask APIs to fetch job updates.  
Displays:

* **Overall Summary Section**
  + Total Tasks, Completed, Failed, Pending, Running
  + Last Executed Task
  + Next Scheduled Task
* **Detailed Job Table**
  + Process Name, Status, Start/End Time, Next Execution
  + Progress bars to visualize execution percentage
* **Auto-refresh mechanism** every few seconds/minutes for real-time updates.

### Step 6: User Interaction

The user (non-Blue Prism operator) accesses the dashboard via a \*\*browser\*\*.  
No need to open Blue Prism Control Room.  
Can track all job statuses and details in real-time.  
Optionally can drill down per task or export data later.

## 4. Technologies Used (By Layer)

| Layer | Technology | Purpose |
| --- | --- | --- |
| **Blue Prism Layer** | Blue Prism Runtime & Control Room | Job execution & scheduling |
| **Data Access Layer** | Python, `requests` / `pyodbc` | Connect to Blue Prism via API or SQL |
| **Storage Layer** | SQLite (Lightweight DB) | Store job history & execution logs |
| **Backend Layer** | Flask (Python Web Framework) | API + Data Aggregation + Web Server |
| **Frontend Layer** | HTML, CSS, JavaScript | Dashboard UI, live updates |
| **Optional Enhancements** | Chart.js / Plotly (Visualization) | Show trends and charts |
| **Environment** | Runs locally or on small VM/server | Self-contained app |

## 5. Data Flow Summary

| Step | Source | Action | Destination |
| --- | --- | --- | --- |
| 1 | Blue Prism | Job execution logs generated | BP DB / API |
| 2 | Python Connector | Fetch job data | Python memory |
| 3 | DB Helper | Save logs | SQLite database |
| 4 | Flask App | Compute summary | Flask API |
| 5 | Frontend | Fetch `/api/status` | Web UI display |
| 6 | Browser | User views dashboard | Visual progress |

## 6. Key Metrics Shown in Dashboard

| Metric | Description |
| --- | --- |
| **Total Tasks** | Total active jobs in system |
| **Completed Count** | Number of successfully completed jobs |
| **Running Count** | Jobs currently in execution |
| **Pending Count** | Jobs scheduled but not yet started |
| **Failed Count** | Jobs that ended in error/failure |
| **Last Executed Task** | Most recent job that completed successfully |
| **Next Scheduled Task** | Next upcoming scheduled job |
| **Per Task Progress** | % completion shown as visual progress bar |
| **Start/End Time** | Execution duration visibility |
| **Next Execution Time** | When job will run next (if recurring) |

## 7. Benefits of This Architecture

| Benefit | Description |
| --- | --- |
| **Independent Monitoring** | Users can track Blue Prism activity without logging into Control Room |
| **Lightweight & Fast** | Uses Flask + SQLite (no heavy servers) |
| **Historical Logs** | SQLite retains execution history for analysis |
| **Customizable UI** | Frontend can be styled or integrated into any internal portal |
| **Scalable Design** | Can easily switch SQLite → PostgreSQL or REST → DB input |
| **Extensible** | Later can add alerting, charts, authentication, or reporting |

## 8. Possible Future Enhancements

| Enhancement | Description |
| --- | --- |
| **Email/Teams Alerts** | Notify when a job fails or exceeds threshold |
| **Charts & Trends** | Visualize execution success rate over time |
| **Authentication Layer** | Secure access for authorized users |
| **API Pagination** | For large datasets |
| **Historical Dashboard** | View stats for past days/weeks |
| **Integration with FastAPI** | For performance and async data fetching |
| **Docker Deployment** | Containerize the tracker app for easy deployment |

## 9. End-to-End Flow Summary

1. Blue Prism executes jobs
2. Python connector fetches job/session info
3. Data logged in SQLite
4. Flask backend aggregates summary
5. HTML+JS frontend requests `/api/status`
6. Dashboard displays:  
       - Overall stats (counts + last/next jobs)  
       - Table with per-job progress bars
7. User views real-time progress