Dishtha Yantra UI Framework

Requirements Specification

*Version 1.0
Web Application for DAG Compute Server Management*

# 1. Executive Summary

The Dishtha Yantra UI Framework is a comprehensive web-based management interface for the DAG Compute Server. Built with Flask and Bootstrap 5, it provides a responsive, secure, and feature-rich platform for managing directed acyclic graphs (DAGs), monitoring system state, and controlling cache operations.

This document specifies the requirements for the web application, focusing on authentication and authorization mechanisms, user interface features, and integration with the DAG compute server backend.

# 2. System Architecture

## 2.1 Technology Stack

The UI Framework is built on the following technologies:

* **Backend Framework:** Flask (Python web framework)
* **Frontend Framework:** Bootstrap 5 (responsive CSS framework)
* **JavaScript Libraries:** jQuery 3.6.0
* **Icons:** Bootstrap Icons 1.7.2
* **Templating Engine:** Jinja2 (Flask's default template engine)

## 2.2 Application Structure

The application follows a Model-View-Controller (MVC) architecture pattern with clear separation of concerns:

| **Component** | **Responsibility** |
| --- | --- |
| **app.py** | Main Flask application with route definitions, authentication decorators, and business logic |
| **templates/** | Jinja2 templates for rendering HTML pages (base.html, dashboard.html, etc.) |
| **dag\_server** | Backend DAG compute server managing graph execution and state |
| **redis\_cache** | In-memory cache system for data persistence and pub/sub messaging |

# 3. Authentication and Authorization

## 3.1 User Authentication

### 3.1.1 User Credential Storage

User credentials are stored in a JSON file (users.json) with the following structure:

{

"username": {

"password": "plaintext\_password",

"full\_name": "User Full Name",

"roles": ["admin", "user", "operator"]

}

}

### 3.1.2 Login Process

1. User enters username and password on login page
2. Credentials are verified against users.json file
3. Upon successful authentication, Flask session stores: username, full\_name, roles[]
4. User is redirected to dashboard with flash message confirmation
5. Failed authentication displays error message and remains on login page

### 3.1.3 Session Management

Session requirements:

* **Secret Key:** Configurable via SECRET\_KEY environment variable
* **Session Data:** Username, full name, and roles array
* **Session Persistence:** Until user explicitly logs out or browser session ends
* **Logout:** Clears all session data and redirects to login

## 3.2 Role-Based Access Control

### 3.2.1 Role Definitions

| **Role** | **Permissions** | **Restrictions** |
| --- | --- | --- |
| **user** | View dashboard, DAG details, DAG state, cache statistics | Cannot create/delete DAGs, cannot modify cache, cannot publish messages |
| **operator** | All user permissions plus start/stop DAG operations | Cannot create/delete DAGs, limited cache management |
| **admin** | Full system access including create/delete DAGs, full cache management, message publishing, system configuration | No restrictions |

**Implementation Notes:**

* Users can have multiple roles simultaneously
* Role checks use Python decorators (@login\_required, @admin\_required)
* Admin badge displayed in navigation bar for users with admin role

### 3.2.2 Route Protection

Routes are protected using decorator functions:

**@login\_required**

Ensures user is authenticated before accessing route. Redirects to login page if session is not active.

**@admin\_required**

Requires both authentication and admin role. Displays error message and redirects to dashboard if user lacks admin privileges.

# 4. User Interface Features

## 4.1 Common UI Components

### 4.1.1 Navigation Bar

Fixed-top navigation bar with the following elements:

* **Brand Logo:** DishtaYantra Compute Server with diagram icon
* **Navigation Links:** Dashboard, Cache (visible when authenticated)
* **User Info:** Displays full name or username with person icon
* **Admin Badge:** Warning badge displayed for users with admin role
* **Logout Link:** Sign out button with icon

### 4.1.2 Flash Messages

Fixed-position alert system for user notifications:

* **Position:** Top-right corner, below navigation bar
* **Types:** Success (green), Error (red), Warning (yellow), Info (blue)
* **Auto-dismiss:** User can dismiss with close button
* **Categories:** success, error, warning, info

### 4.1.3 Footer

Page footer contains:

* **Copyright:** © 2025-2030 Ashutosh Sinha. All rights reserved.
* **Contact Email:** ajsinha@gmail.com
* **GitHub Repository:** Link to https://github.com/ajsinha/dishtayantra

### 4.1.4 Pagination Controls

Standardized pagination for all data tables:

* **Page Size Selector:** 10, 50, 100, or All rows options
* **Navigation Buttons:** First, Previous, Next, Last page controls
* **Page Info:** Displays 'Showing X to Y of Z entries' with filtering context
* **Client-Side Processing:** All pagination handled via JavaScript

### 4.1.5 Auto-Refresh Functionality

Real-time data updates with user control:

* **Toggle Switch:** Enable/disable auto-refresh
* **Countdown Timer:** Displays seconds until next refresh
* **Refresh Intervals:** 30 seconds for dashboard, 5 seconds for DAG state
* **Visual Feedback:** Badge color changes based on active/inactive state

## 4.2 Dashboard Page

Main landing page after authentication, providing comprehensive DAG management interface.

### 4.2.1 Server Status Display

* **Primary/Standby Badge:** Green badge for PRIMARY, gray badge for STANDBY
* **Header Clock:** Real-time HH:MM display updated every minute
* **Create DAG Button:** Visible only for admin users

### 4.2.2 DAG Table Features

| **Feature** | **Description** |
| --- | --- |
| Name Display | Shows DAG name with clone indicator if auto-cloned, including parent DAG reference |
| Status Badge | Running (green), Stopped (gray), Suspended (yellow) with color-coded badges |
| Time Window | Displays start time with duration badge (e.g., 09:00 + 8h), shows end time and default -5m indicator |
| Window Status | Active/Inactive indicator with colored dot (green for in-window, red for outside) |
| Node Count | Total number of nodes in the DAG |
| AutoClone Status | Enable/Disable toggle for automatic DAG cloning at start\_time |
| Actions | Details, State, Start, Stop, Resume, Suspend, Clone, Delete buttons (admin-only operations protected) |

**Table Enhancements:**

* **Search:** Real-time filtering by DAG name
* **Sorting:** Click column headers to sort (name, status, window status, nodes)
* **Pagination:** Configurable rows per page with navigation controls
* **Row Highlighting:** Auto-cloned DAGs shown with blue background tint

### 4.2.3 DAG Operations

| **Operation** | **Description** | **Access Control** |
| --- | --- | --- |
| **Details** | View comprehensive DAG configuration, nodes, subscribers, publishers, and edges | All authenticated users |
| **State** | Monitor real-time node input/output, dirty status, calculation counts, and errors | All authenticated users |
| **Start** | Initiate DAG execution and computation loop | Admin and operator roles |
| **Stop** | Halt DAG execution gracefully | Admin and operator roles |
| **Resume** | Continue DAG execution from suspended state | Admin and operator roles |
| **Suspend** | Temporarily pause DAG execution without stopping | Admin and operator roles |
| **Clone** | Create copy with optional time window modifications | Admin only |
| **Delete** | Permanently remove DAG with confirmation dialog | Admin only |

## 4.3 Create DAG Page

### 4.3.1 Configuration Upload

* **File Upload:** JSON file selector with .json file type filter
* **Validation:** Server-side JSON validation and structure verification
* **Requirements Display:** Information box showing mandatory fields
* **Example Configuration:** Expandable code snippet showing complete JSON structure

### 4.3.2 Configuration Requirements

Mandatory fields in DAG configuration:

* **name:** Unique alphanumeric identifier with underscores
* **subscribers:** Array of data subscriber configurations
* **publishers:** Array of data publisher configurations
* **nodes:** Array of node definitions with types and configurations
* **edges:** Array of directed connections between nodes

Optional fields:

* **start\_time:** HHMM format (e.g., 0900 for 9:00 AM)
* **duration:** Format: Xh, Ym, or XhYm (e.g., 1h30m)
* **calculators:** Array of calculator configurations
* **transformers:** Array of data transformation configurations

## 4.4 Clone DAG Page

### 4.4.1 Cloning Configuration

Features for creating DAG copies with time window modifications:

* **Original DAG Info:** Display box showing source DAG name and current time window
* **Start Time Field:** HH:MM or HHMM format input with validation
* **Duration Field:** Pattern-validated input (Xh, Ym, XhYm format)
* **Generated Name:** Automatic timestamp suffix (original\_name\_YYYYMMDDHHMMSS)
* **Important Notes Section:** Comprehensive help text explaining perpetual running, default duration, and format requirements

### 4.4.2 Time Window Configuration

| **Start Time** | **Duration** | **End Time** | **Behavior** |
| --- | --- | --- | --- |
| 0900 | 8h | 17:00 | Business hours (9 AM to 5 PM) |
| 0600 | 1h30m | 07:30 | Morning batch job |
| 0900 | *empty* | 08:55 | Default -5 min duration |
| *empty* | *empty* | - | Always active (24/7) |

**Duration Format Reference:**

* 15m, 30m, 45m - Minutes only
* 1h, 2h, 4h, 8h, 12h - Hours only
* 1h15m, 1h30m, 2h30m, 7h30m - Mixed hours and minutes

## 4.5 DAG Details Page

### 4.5.1 Node Information Display

Topologically sorted node table with comprehensive details:

* **Node Name:** Clickable anchor links for quick navigation
* **Node Type:** Badge-styled display (SubscriptionNode, CalculationNode, PublicationNode, etc.)
* **Dependencies:** List of parent nodes with clickable links
* **Last Calculation:** ISO timestamp or 'Never' indicator
* **Errors:** Expandable error count button showing detailed error list with timestamps
* **Configuration:** Expandable JSON viewer for node configuration

### 4.5.2 Subscribers Section

* **Name:** Subscriber identifier
* **Source:** Connection string (mem://, kafka://, redischannel://, activemq://)
* **Queue Depth:** Current/Max depth display
* **Last Receive:** Timestamp or 'Never' indicator
* **Publish Action:** Admin-only button for supported protocols (mem, kafka, redischannel, activemq)

### 4.5.3 Publishers Section

* **Name:** Publisher identifier
* **Destination:** Target connection string
* **Last Publish:** Timestamp or 'Never' indicator

### 4.5.4 Edges Section

* **Edge Name:** Auto-generated as 'from\_node\_to\_to\_node'
* **From/To Nodes:** Source and destination node names with clickable links
* **Transformer:** Optional data transformer name or 'None'

## 4.6 DAG State Page

### 4.6.1 Real-Time State Monitoring

Live monitoring of node execution state with 5-second auto-refresh:

* **Node Name:** Identifier for each node in topological order
* **Status Badge:** Dirty (yellow) or Clean (green) indicating computation state
* **Calculation Count:** Total number of computations executed
* **Last Calculation:** ISO timestamp of most recent computation
* **Input State:** Expandable JSON viewer for current input data
* **Output State:** Expandable JSON viewer for current output data
* **Errors:** Count and expandable list showing recent errors with timestamps

### 4.6.2 Auto-Refresh Controls

* **Refresh Toggle:** Enable/disable auto-refresh switch
* **Countdown Timer:** 5-second countdown badge showing time to next refresh
* **Manual Refresh:** Button to immediately reload page data
* **Visual Feedback:** Badge color changes (blue for active, gray for disabled)

## 4.7 Publish Message Page

### 4.7.1 Message Publishing Interface

Admin-only interface for publishing data to DAG subscribers:

* **Subscriber Info Panel:** Shows name, source, queue depth, and last receive timestamp
* **Message Editor:** Monospace textarea for JSON message input (15 rows)
* **JSON Validation:** Client-side validation button with syntax error reporting
* **Publish Action:** AJAX-based submission with loading state indicator
* **Clear Button:** Confirmation dialog before clearing message
* **Tips Panel:** Help text with publishing guidelines

### 4.7.2 Validation and Feedback

* **Pre-publish Validation:** JSON.parse() check before sending
* **Success Message:** Green alert confirming successful publication
* **Error Display:** Red alert showing specific error message
* **Loading State:** Blue alert with hourglass icon during submission

## 4.8 Cache Management Page

### 4.8.1 Cache Statistics Dashboard

Gradient-styled statistics panel displaying:

* **Total Keys:** Count of all cache entries
* **Keys with TTL:** Count of expiring entries
* **Keys without TTL:** Count of persistent entries
* **Last Dump:** Timestamp of last cache persistence operation
* **Dump Interval:** Configured persistence frequency
* **Type Breakdown:** Distribution of string, hash, list, and other types

### 4.8.2 Query Interface

* **Pattern Search:** Wildcard support (\* for any characters)
* **Common Patterns:** user:\*, \*:session, \*:cache, specific\_key
* **Results Display:** Paginated table with key, value preview, type, and TTL
* **Pagination:** 10, 50, 100, or All entries per page

### 4.8.3 Admin Operations

| **Operation** | **Description** |
| --- | --- |
| **Create Entry** | Modal dialog for adding new key-value pairs with optional TTL (seconds) |
| **View Full Value** | Modal dialog showing complete value with copy-to-clipboard functionality |
| **Update TTL** | Modal dialog for modifying expiration time (-1 for no expiry, positive integer for seconds) |
| **Delete Entry** | Confirmation dialog before permanent deletion |
| **Download Cache** | Export entire cache as JSON file with timestamp (cache\_export\_YYYYMMDD\_HHMMSS.json) |
| **Dump Now** | Manually trigger cache persistence to disk immediately |
| **Clear Cache** | Remove all entries with strong confirmation dialog (flushall operation) |

# 5. Server Integration

## 5.1 DAG Server Communication

### 5.1.1 Server Initialization

Flask application initializes DAGComputeServer with:

* **Configuration Folder:** Path to JSON configuration files (default: ./config/dags)
* **ZooKeeper Hosts:** Coordination service connection string (default: localhost:2181)
* **Environment Variables:** DAG\_CONFIG\_FOLDER, ZOOKEEPER\_HOSTS, USERS\_FILE, SECRET\_KEY

### 5.1.2 API Methods

| **Method** | **Purpose** |
| --- | --- |
| get\_server\_status() | Returns primary/standby status for HA display |
| list\_dags() | Returns array of DAG metadata for dashboard display |
| details(dag\_name) | Returns comprehensive DAG details including nodes, edges, subscribers, publishers |
| create\_dag(config) | Creates new DAG from JSON configuration dictionary |
| start\_dag(dag\_name) | Initiates DAG execution loop |
| stop\_dag(dag\_name) | Gracefully stops DAG execution |
| suspend\_dag(dag\_name) | Temporarily pauses DAG execution without stopping |
| resume\_dag(dag\_name) | Resumes suspended DAG execution |
| clone\_dag(...) | Creates DAG copy with optional time window modifications |
| delete\_dag(dag\_name) | Permanently removes DAG and configuration |
| enable\_autoclone(...) | Activates automatic DAG cloning at start\_time |
| disable\_autoclone(...) | Deactivates automatic DAG cloning |
| publish\_to\_subscriber(...) | Sends data message to specified subscriber queue |

## 5.2 Cache System Integration

### 5.2.1 InMemoryRedisClone Interface

Redis-compatible in-memory cache with persistence:

* **Data Operations:** get, set, delete, exists, keys, type
* **Expiration:** expire (set TTL), ttl (get TTL), persist (remove TTL)
* **Persistence:** dump\_to\_file, load\_from\_file, get\_dump\_info
* **Admin Operations:** flushall (clear all entries)

### 5.2.2 RESTful Cache API Endpoints

| **Endpoint** | **Method** | **Purpose** |
| --- | --- | --- |
| /cache/api/query | GET | Search cache with pattern, pagination, returns JSON |
| /cache/api/create | POST | Create new cache entry with optional TTL (admin) |
| /cache/api/delete | DELETE | Remove cache entry by key (admin) |
| /cache/api/ttl | PUT | Update TTL for existing entry (admin) |
| /cache/api/clear | POST | Clear entire cache (flushall, admin only) |
| /cache/api/download | GET | Export cache as JSON file (admin) |
| /cache/api/stats | GET | Get cache statistics and dump info |
| /cache/api/dump/trigger | POST | Manually trigger cache dump to disk (admin) |

# 6. Technical Requirements

## 6.1 Browser Compatibility

* Modern browsers: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
* JavaScript: ES6+ features required
* Responsive design: Minimum viewport width 320px, optimal 1024px+

## 6.2 Performance Requirements

* **Page Load Time:** < 3 seconds for initial dashboard load
* **Auto-Refresh Response:** < 1 second for page reload
* **AJAX Operations:** < 500ms response time for cache API calls
* **Table Pagination:** Client-side processing for up to 1000 rows
* **Search/Filter:** < 200ms response time for user input

## 6.3 Security Requirements

* **Session Security:** Flask secure sessions with configurable secret key
* **Password Storage:** Currently plaintext (future: bcrypt/scrypt hashing required)
* **CSRF Protection:** Flask-WTF CSRF tokens for state-changing operations
* **Input Validation:** Server-side validation for all user inputs
* **HTTPS:** Recommended for production deployment

## 6.4 Logging and Monitoring

* **Application Logs:** Python logging module with INFO level
* **Log Format:** Timestamp - Module - Level - Message
* **Error Tracking:** Exception logging with stack traces
* **Admin Actions:** Logged with username and timestamp

# 7. Deployment Requirements

## 7.1 Environment Configuration

| **Variable** | **Purpose** |
| --- | --- |
| SECRET\_KEY | Flask session encryption key (default: dagserver\_secret\_key\_change\_me) |
| DAG\_CONFIG\_FOLDER | Path to DAG JSON configurations (default: ./config/dags) |
| ZOOKEEPER\_HOSTS | ZooKeeper connection string (default: localhost:2181) |
| USERS\_FILE | Path to users.json file (default: ./config/users.json) |

## 7.2 Directory Structure

./config/dags/ # DAG JSON configurations

./config/users.json # User credentials

./logs/ # Application logs

./templates/ # Jinja2 HTML templates

./static/ # CSS, JS, images (if any)

## 7.3 Production Deployment

* **WSGI Server:** Gunicorn or uWSGI recommended
* **Reverse Proxy:** Nginx or Apache for static files and SSL termination
* **SSL Certificate:** Let's Encrypt or commercial certificate
* **Port Configuration:** Default 5000 (Flask), recommend 443 (HTTPS) via reverse proxy
* **Process Management:** Systemd or Supervisor for automatic restart

# 8. Future Enhancements

## 8.1 Authentication Improvements

* Password hashing with bcrypt/scrypt
* OAuth2/SAML integration for enterprise SSO
* Multi-factor authentication support
* API token-based authentication for programmatic access

## 8.2 UI Enhancements

* DAG visualization with interactive graph diagrams
* Real-time WebSocket updates for state monitoring
* Advanced filtering and query builder for cache search
* Export DAG configurations as downloadable JSON
* Dark mode theme option

## 8.3 Monitoring and Analytics

* Performance metrics dashboard with charts
* Historical execution logs with time-series graphs
* Alert configuration for DAG failures
* Email/Slack notifications for critical events

# 9. Conclusion

The Dishtha Yantra UI Framework provides a comprehensive, secure, and user-friendly interface for managing DAG computation workloads. The framework successfully integrates:

* **Robust Authentication:** Role-based access control ensuring secure operations
* **Intuitive Interface:** Bootstrap-based responsive design with real-time updates
* **Comprehensive Management:** Full lifecycle control of DAGs and cache operations
* **Powerful Monitoring:** Real-time state inspection with auto-refresh capabilities
* **Seamless Integration:** Direct communication with DAG compute server and cache system

The modular architecture allows for easy extension and customization, while the clear separation of concerns ensures maintainability and scalability for future enhancements.

*— End of Document —*