# Magic Tool - Complete Documentation

## Overview

The **Magic Tool** (accessible via `/magic-created` route in the admin panel) is an automated notification tracking and management system that monitors multiple government and organizational websites for new updates, results, and announcements. It provides real-time notifications, intelligent filtering, and streamlined workflows for processing and publishing updates.

### Key Features

- **Automated Web Scraping**: Monitors configured sources (HTML/RSS) at regular intervals
- **Real-time Updates**: Socket.io integration for live notification delivery
- **Smart Filtering**: Separates result notifications from general updates
- **Draft Management**: Review, edit, and batch upload notifications
- **Source Management**: CRUD operations for managing monitored sources
- **Excel Export**: Download today's notifications for offline processing

---

## End-to-End Process Flow

This section explains the complete lifecycle of the Magic Tool system, from startup to notification delivery.

### System Initialization (Server Startup)

```
flowchart TD
    A[Server Starts] --> B[Load Environment Variables]
    B --> C[Connect to MongoDB]
    C --> D[Initialize Express App]
    D --> E[Create HTTP Server]
    E --> F[Initialize Socket.io Server]
    F --> G[Setup CORS for Socket.io]
    G --> H[Register Socket Event Handlers]
    H --> I[Setup Cron Jobs]
    I --> J{Cron Schedule}
    J -->|Every 2 min| K[Monitor Sources Job]
    J -->|Daily 00:00| L[Vector Embedding Job]
    J -->|Every 30 min| M[Publish Scheduled PDFs]
    J -->|Daily 00:00| N[Report Generation Job]
    K --> O[Server Ready]
    L --> O
    M --> O
    N --> O
    O --> P[Listening on Port 5000]
```

**Step-by-Step Breakdown**:

1. **Server Initialization** ( `adminserver/index.js` ):
    - Environment variables loaded from `.env`
    - MongoDB connection established
    - Express app factory creates the application

- HTTP server wraps Express app

2. **Socket.io Setup**:
   - Socket.io server created with CORS configuration
   - Allowed origins include localhost and production domains
   - Connection event handler registered
   - `subscribe-source` event listener added
   - Global `io` object made available to jobs
3. **Cron Jobs Registration**:
   - **Monitor Sources**: `*/2 * * * *` (every 2 minutes)
   - **Vector Embedding**: `0 0 * * *` (daily at midnight)
   - **Publish Scheduled PDFs**: `*/30 * * * *` (every 30 minutes)
   - **Report Generation**: `0 0 * * *` (daily at midnight)
4. **Server Ready**: Listening on port 5000 (or `process.env.PORT`)

---

## Complete Notification Lifecycle

This is the **complete journey** of a notification from discovery to display in the admin panel.

```
flowchart TD
    Start([System Running]) --> Cron{Cron Trigger<br/>Every 2 min}
    Cron --> FetchSources[Fetch Active Sources<br/>from MongoDB]
    FetchSources --> LoopSources{For Each<br/>Source}

    LoopSources --> CheckInterval{Enough time<br/>passed?}
    CheckInterval -->|No| LoopSources
    CheckInterval -->|Yes| DetermineType{Source Type?}

    DetermineType -->|HTML| ScrapeHTML[HTTP GET HTML Page]
    DetermineType -->|RSS| ScrapeRSS[HTTP GET RSS Feed]

    ScrapeHTML --> ParseHTML[Parse with Cheerio<br/>Extract links using selector]
    ScrapeRSS --> ParseRSS[Parse with rss-parser<br/>Extract items]

    ParseHTML --> Items[Items Array]
    ParseRSS --> Items

    Items --> CheckItems{Items found?}
    CheckItems -->|No| UpdateTime[Update lastCheckedAt]
    CheckItems -->|Yes| ProcessItems[Process Each Item]

    ProcessItems --> CreateHash[Create SHA-256 Hash<br/>sourceCode|title|link]
    CreateHash --> Upsert[MongoDB Upsert<br/>bulkWrite operation]

    Upsert --> CheckNew{New documents<br/>inserted?}
    CheckNew -->|No| UpdateTime
    CheckNew -->|Yes| FilterTime[Filter Last 4 Days]

    FilterTime --> CheckFiltered{Fresh docs<br/>exist?}
    CheckFiltered -->|No| UpdateTime
    CheckFiltered -->|Yes| EmitSocket[Emit Socket.io Events]

    EmitSocket --> RoomEmit[Emit to room sourceCode]
    EmitSocket --> GlobalEmit[Emit global event]

    RoomEmit --> UpdateTime
    GlobalEmit --> UpdateTime
    UpdateTime --> LoopSources

    LoopSources -->|All Done| End([Wait for Next Cron])

    RoomEmit -.->|WebSocket| ClientReceive[Client Receives Event]
    GlobalEmit -.->|WebSocket| ClientReceive

    ClientReceive --> ClientCheck{Already seen?}
    ClientCheck -->|Yes| ClientSkip[Skip duplicate]
    ClientCheck -->|No| ClientAdd[Add to state]
    ClientAdd --> ClientSort[Sort by timestamp]
    ClientSort --> ClientRender[Re-render UI]
    ClientRender --> ShowIndicator[Show blue dot indicator]
```

---

### Detailed Phase Breakdown

#### Phase 1: System Startup

**What Happens**:

1. Node.js server starts
2. MongoDB connection established
3. Express routes registered
4. Socket.io server initialized
5. Cron jobs scheduled
6. Server begins listening

**Key Files**:

- `adminserver/index.js` - Main entry point
- `adminserver/app.js` - Express app factory
- `adminserver/config/db.js` - Database connection

**Timeline**: ~2-5 seconds

---

#### Phase 2: Admin Opens Magic Tool

**User Action**: Admin navigates to `/magic-created` in browser

**What Happens**:

```
sequenceDiagram
    autonumber
    participant Browser
    participant React as TrackNewUpdates
    participant API as Backend API
    participant DB as MongoDB
    participant Socket as Socket.io

    Browser->>React: Navigate to /magic-created
    React->>React: Component mounts

    React->>API: GET /api/sources
    API->>DB: Find all sources
    DB-->>API: sources[]
    API-->>React: Response with sources
    React->>React: setSources(sources)
    React->>React: Filter active sources

    par Fetch notifications for each source
        React->>API: GET /api/notifications?sourceCode=SOURCE1&limit=60
        React->>API: GET /api/notifications?sourceCode=SOURCE2&limit=60
        React->>API: GET /api/notifications?sourceCode=SOURCE3&limit=60
    end

    API->>DB: Find notifications by sourceCode
    DB-->>API: notifications[]
    API-->>React: Response with notifications

    React->>React: setNotificationsBySource({...})
    React->>React: Initialize seenRef for deduplication
    React->>React: Expand first 2 sources

    React->>Socket: Connect to Socket.io server
    Socket-->>React: Connection established

    loop For each active source
        React->>Socket: emit("subscribe-source", sourceCode)
        Socket->>Socket: socket.join(sourceCode)
    end

    React->>Socket: on("new-notifications", handler)
    React->>Browser: Render UI with initial data
```

**Timeline**: ~1-3 seconds (depending on number of sources)

---

#### Phase 3: Automated Scraping (Every 2 Minutes)

**Trigger**: Cron job executes `monitorAllSources(io)`

**What Happens**:

```
sequenceDiagram
    autonumber
    participant Cron
    participant Job as monitorAllSources
    participant DB as MongoDB
    participant Scraper as fetchItemsForSource
    participant Website as External Website
    participant NotifSvc as saveNewNotifications
    participant Socket as Socket.io

    Cron->>Job: Trigger (every 2 min)
    Job->>DB: Source.find({ isActive: true })
    DB-->>Job: activeSources[]

    loop For each source
        Job->>Job: Check if intervalMinutes passed

        alt Interval not passed
            Job->>Job: Skip this source
        else Interval passed
            Job->>Scraper: fetchItemsForSource(source)

            alt source.type === "rss"
                Scraper->>Website: axios.get(notificationUrl)
                Website-->>Scraper: XML response
                Scraper->>Scraper: rssParser.parseString(xml)
                Scraper->>Scraper: Map items to standard format
            else source.type === "html"
                Scraper->>Website: axios.get(notificationUrl)
                Website-->>Scraper: HTML response
                Scraper->>Scraper: cheerio.load(html)
                Scraper->>Scraper: $(selector).find("a")
                Scraper->>Scraper: Extract title & href
                Scraper->>Scraper: Resolve relative URLs
            end

            Scraper-->>Job: items[]

            alt items.length > 0
                Job->>NotifSvc: saveNewNotifications(source, items)

                loop For each item
                    NotifSvc->>NotifSvc: Create itemHash (SHA-256)
                    NotifSvc->>NotifSvc: Build upsert operation
                end

                NotifSvc->>DB: Notification.bulkWrite(operations)
                DB-->>NotifSvc: { upsertedIds: {...} }

                NotifSvc->>DB: Find newly inserted docs
                DB-->>NotifSvc: newDocs[]
                NotifSvc-->>Job: newDocs[]

                alt newDocs.length > 0
                    Job->>Job: Filter last 4 days (getFreshCutoff)

                    alt freshDocs.length > 0
```

```
                        Job->>Socket: io.to(sourceCode).emit("new-notifications",
freshDocs)
                        Job->>Socket: io.emit("new-notifications-global", freshDocs)
                        Socket-->>Socket: Broadcast to subscribed clients
                    end
                end
            end

            Job->>DB: Update source.lastCheckedAt = new Date()
        end
    end

    Job->>Job: Log "monitorAllSources done"
```

**Timeline**: ~5-30 seconds (depending on number of sources and website response times)

---

**Phase 4: Real-time Notification Delivery**

**Trigger**: Socket.io emits `new-notifications` event

**What Happens**:

```
sequenceDiagram
    autonumber
    participant Job as Monitor Job
    participant Socket as Socket.io Server
    participant Client as TrackNewUpdates Component
    participant State as React State
    participant UI as Browser UI

    Job->>Socket: io.to(sourceCode).emit("new-notifications", freshDocs)
    Socket->>Client: Broadcast to room subscribers

    Client->>Client: Receive event with incoming docs

    loop For each doc in incoming
        Client->>Client: Extract sourceCode from doc
        Client->>Client: Create key = makeKey(doc)

        alt seenRef[sourceCode].has(key)
            Client->>Client: Skip (already seen)
        else Not seen
            Client->>Client: seenRef[sourceCode].add(key)
            Client->>Client: newSinceOpenRef.add(key)
            Client->>State: Add doc to notificationsBySource[sourceCode]
            Client->>State: Sort by timestamp (newest first)
            Client->>State: Limit to 400 notifications
        end
    end

    Client->>State: setNewSinceOpenCount(newSinceOpenRef.size)
    Client->>State: setNotificationsBySource(updated)
    State->>UI: Trigger re-render
    UI->>UI: Display new notification
    UI->>UI: Show blue dot indicator
    UI->>UI: Update "New since open" counter
    UI->>UI: Update "Today" counter
```

**Timeline**: ~100-500ms (instant from user perspective)

---

#### Phase 5: Admin Processes Result Notifications

**User Action**: Admin clicks "Results Drafts" button

**What Happens**:

```
sequenceDiagram
    autonumber
    participant Admin
    participant Track as TrackNewUpdates
    participant Panel as ResultNotificationsPanel
    participant Modal as Edit Modal
    participant API as Backend API

    Admin->>Track: Click "Results Drafts" button
    Track->>Track: setResultDrawerOpen(true)
    Track->>Panel: Render with resultNotifications

    Panel->>Panel: Filter isResultNotification(n)
    Panel->>Panel: Build draft payloads
    Panel->>Admin: Display result notifications

    Admin->>Panel: Select notifications (checkboxes)
    Panel->>Panel: Update selected Set

    Admin->>Panel: Click "Edit & Upload (N)"
    Panel->>Modal: openEditModal(draftsSelected)
    Modal->>Modal: setEditDrafts(deepClone(drafts))
    Modal->>Admin: Display edit form

    Admin->>Modal: Edit title, link, resultDate, etc.
    Modal->>Modal: updateDraftField(idx, field, value)

    Admin->>Modal: Click "Final Upload"
    Modal->>Modal: validateDrafts(editDrafts)

    alt Validation fails
        Modal->>Admin: Show error message
    else Validation passes
        Modal->>API: POST /api/results/import { rows: editDrafts }
        API->>API: Process and save results
        API-->>Modal: 200 OK
        Modal->>Admin: Success alert
        Modal->>Modal: closeEditModal()
        Modal->>Panel: closeDraftModal()
        Panel->>Panel: clearAll()
    end
```

**Timeline**: Variable (depends on admin editing time)

---

### Critical Timing & Intervals

| Event | Frequency | File | Description |
|---|---|---|---|
| **Cron Trigger** | Every 2 minutes | `adminserver/index.js` | Triggers monitoring job |
| **Source Scraping** | Per `intervalMinutes` | `adminserver/Jobs/monitorSources.job.js` | Individual source check interval |
| **Socket Broadcast** | Immediate | `adminserver/Jobs/monitorSources.job.js` | When new notifications found |
| **Client Polling** | N/A (WebSocket) | `gyapakAdminPanel/src/Pages/TrackNewUpdates.jsx` | Real-time via Socket.io |
| **UI Re-render** | On state change | React | Automatic React re-rendering |

---

### Data Persistence Points

**Where data is saved**:

1. **Source Configuration**:
   - Saved to: `sources` collection in MongoDB
   - When: Admin creates/updates source via ManageSources
   - Persists: Permanently (until deleted)
2. **Notifications**:
   - Saved to: `notifications` collection in MongoDB
   - When: Scraper finds new items
   - Persists: Permanently (manual cleanup needed)
3. **Last Checked Timestamp**:
   - Saved to: `source.lastCheckedAt` field
   - When: After each scraping attempt
   - Persists: Updated every scrape
4. **Client State**:
   - Saved to: React component state (memory only)
   - When: Initial load + socket updates

- Persists: Only while page is open (lost on refresh)

5. **Seen References**:
   - Saved to: `seenRef` (React ref, memory only)
   - When: Notifications loaded/received
   - Persists: Only while page is open

---

## Error Handling & Recovery

**What happens when things fail**:

1. **Website is down**:
   - Scraper catches error
   - Logs error message
   - Returns empty array
   - Continues with next source
   - Retries on next cron cycle (2 min)
2. **MongoDB connection lost**:
   - Server crashes
   - Needs manual restart
   - Cron jobs stop
   - Socket.io clients disconnect
   - Auto-reconnect on server restart
3. **Socket.io disconnect**:
   - Client attempts reconnection
   - UI continues working (no real-time updates)
   - Data still available from initial load
   - Reconnects automatically when server available
4. **Duplicate notifications**:
   - MongoDB unique indexes prevent insertion
   - `bulkWrite` with `ordered: false` continues
   - Only new notifications inserted
   - No error thrown to user
5. **Invalid CSS selector**:
   - Scraper finds no links
   - Returns empty array
   - Logs "0 items fetched"
   - Admin should check logs and update selector

---

## Complete Timeline Example

**Scenario**: New UPSC notification is published

```
T+0:00  - UPSC publishes new notification on their website
T+0:00  - Magic Tool system is running, waiting for next cron
```

```
T+1:45  - Cron triggers (every 2 min, happened to be 1:45 later)
T+1:46  - Monitor job fetches active sources from DB
T+1:47  - Checks UPSC source, interval passed (last check was 5+ min ago)
T+1:48  - Scraper makes HTTP GET to UPSC notification page
T+1:49  - Website responds with HTML
T+1:50  - Cheerio parses HTML, extracts links using selector
T+1:51  - Found 25 items (including the new one)
T+1:52  - Creates SHA-256 hash for each item
T+1:53  - MongoDB bulkWrite upsert operation
T+1:54  - 1 new document inserted (24 were duplicates)
T+1:55  - Filters last 4 days (new notification is fresh)
T+1:56  - Emits socket event to "CENTRAL_UPSC" room
T+1:57  - Socket.io broadcasts to all subscribed clients
T+1:57  - Admin's browser receives event (if page is open)
T+1:57  - React component checks seenRef (not seen)
T+1:57  - Adds to notificationsBySource state
T+1:57  - Increments newSinceOpenCount
T+1:57  - UI re-renders with blue dot indicator
T+1:57  - Admin sees new notification instantly
```

**Total time from publication to display**: ~1-2 minutes (depends on cron timing)

---

## System Architecture

```
graph TB
    subgraph "Frontend - Admin Panel"
        A[TrackNewUpdates Component]
        B[ManageSources Component]
        C[ResultNotificationsPanel Component]
    end

    subgraph "Backend - Express Server"
        D[Source Controller]
        E[Notification Controller]
        F[Scraper Service]
        G[Notification Service]
    end

    subgraph "Real-time Layer"
        H[Socket.io Server]
        I[Socket.io Client]
    end

    subgraph "Database - MongoDB"
        J[(Source Collection)]
        K[(Notification Collection)]
    end

    subgraph "Automated Jobs"
        L[Monitor Sources Job<br/>Cron: */2 * * * *]
    end

    A -->|GET /api/sources| D
    A -->|GET /api/notifications| E
    A <-->|WebSocket| I
    I <-->|WebSocket| H
    B -->|CRUD /api/sources| D
    C -->|POST /api/results/import| E

    D <-->|Read/Write| J
    E <-->|Read/Write| K

    L -->|Every 2 minutes| F
    F -->|Fetch HTML/RSS| External[External Websites]
    F -->|Save new items| G
    G -->|Bulk insert| K
    G -->|Emit events| H
    H -->|Broadcast| I
```

---

## Database Models

### 1. Source Model

**File**: `adminserver/models/source.model.js`

Stores configuration for each monitored website/source.

```
{
  code: String,              // Unique identifier (e.g., "CENTRAL_UPSC")
  name: String,              // Display name (e.g., "UPSC")
  baseUrl: String,           // Base URL for resolving relative links
  notificationUrl: String,   // URL to scrape for notifications
  type: String,              // "html", "rss", or "json"
  selector: String,          // CSS selector for HTML scraping (optional)
  intervalMinutes: Number,   // Scraping interval (default: 5)
  lastCheckedAt: Date,       // Last scrape timestamp
  isActive: Boolean,         // Enable/disable scraping
  createdAt: Date,
  updatedAt: Date
}
```

**Indexes**:

- `code` : Unique index

### 2. Notification Model

**File**: `adminserver/models/notification.model.js`

Stores individual notifications scraped from sources.

```
{
  sourceId: ObjectId,         // Reference to Source
  sourceCode: String,         // Denormalized for faster queries
  title: String,              // Notification title
  link: String,               // Notification URL
  summary: String,            // Optional description/summary
  rawText: String,            // Raw extracted text
  publishedAt: Date,          // Publication date (from RSS or current date)
  firstSeenAt: Date,          // When first detected by scraper
  itemHash: String,           // SHA-256 hash for deduplication
  createdAt: Date,
  updatedAt: Date
}
```

**Indexes**:

- `{ sourceCode: 1, link: 1 }` : Unique compound index
- `{ sourceCode: 1, itemHash: 1 }` : Unique compound index
- `sourceCode` : Regular index for filtering

---

## Backend Components

### 1. Source Controller

**File**: `adminserver/controllers/source.controller.js`

Handles CRUD operations for sources.

#### Endpoints

| Method | Endpoint | Description |
|---|---|---|
| GET | `/api/sources` | Fetch all sources (sorted by creation date) |
| POST | `/api/sources` | Create a new source |
| PUT | `/api/sources/:id` | Update an existing source |
| DELETE | `/api/sources/:id` | Delete a source (hard delete) |

#### Example: Create Source

```
POST /api/sources
{
  "code": "CENTRAL_UPSC",
  "name": "UPSC",
  "baseUrl": "https://www.upsc.gov.in",
  "notificationUrl": "https://www.upsc.gov.in/recruitment/recruitment-
advertisement",
  "type": "html",
  "selector": ".notice-list a",
  "intervalMinutes": 5
}
```

### 2. Notification Controller

**File**: `adminserver/controllers/notification.controller.js`

Retrieves notifications with optional filtering.

#### Endpoints

| Method | Endpoint | Description |
|---|---|---|
| GET | `/api/notifications` | Fetch notifications (with optional sourceCode filter) |

#### Query Parameters

- `sourceCode` (optional): Filter by specific source
- `limit` (optional): Number of results (default: 50)

#### Example Request

```
GET /api/notifications?sourceCode=CENTRAL_UPSC&limit=60
```

### 3. Scraper Service

**File**: `adminserver/Service/scraper.service.js`

Fetches notifications from external sources based on type.

#### Supported Types

**HTML Scraping**:

- Uses Cheerio to parse HTML
- Extracts links using CSS selectors
- Resolves relative URLs to absolute URLs

**RSS Parsing**:

- Uses `rss-parser` library
- Extracts title, link, summary, and publication date
- Handles malformed RSS feeds

#### Function: `fetchItemsForSource(source)`

```
// Returns array of items:
[
  {
    title: "Notification Title",
    link: "https://example.com/notification",
    summary: "Optional summary text",
    publishedAt: Date
  }
]
```

### 4. Notification Service

**File**: `adminserver/Service/notification.service.js`

Handles deduplication and bulk insertion of notifications.

**Function: `saveNewNotifications(source, items)`**

- Creates SHA-256 hash from `sourceCode|title|link`
- Uses MongoDB `bulkWrite` with `upsert` to avoid duplicates
- Returns only newly inserted documents
- Prevents duplicate notifications using compound unique indexes

---

## Automated Jobs

### Monitor Sources Job

**File**: `adminserver/Jobs/monitorSources.job.js`

**Schedule**: Every 2 minutes ( `*/2 * * * *` )

**Flow**:

```
sequenceDiagram
    participant Cron
    participant Job as Monitor Job
    participant DB as MongoDB
    participant Scraper as Scraper Service
    participant External as External Website
    participant NotifService as Notification Service
    participant Socket as Socket.io

    Cron->>Job: Trigger (every 2 min)
    Job->>DB: Find active sources

    loop For each active source
        Job->>Job: Check interval (skip if too soon)
        Job->>Scraper: fetchItemsForSource(source)
        Scraper->>External: HTTP GET (HTML/RSS)
        External-->>Scraper: Response
        Scraper-->>Job: Parsed items[]

        Job->>NotifService: saveNewNotifications(source, items)
        NotifService->>DB: bulkWrite (upsert)
        DB-->>NotifService: New documents

        alt Has new notifications
            NotifService-->>Job: newDocs[]
            Job->>Job: Filter last 4 days
            Job->>Socket: Emit to room(sourceCode)
            Job->>Socket: Emit global event
        end

        Job->>DB: Update source.lastCheckedAt
    end
```

**Key Logic**:

1. Fetches all active sources from database
2. For each source:
   - Checks if enough time has passed based on `intervalMinutes`
   - Scrapes the source using `fetchItemsForSource`
   - Saves new notifications using `saveNewNotifications`
   - Filters notifications from last 4 days
   - Emits socket events to connected clients
   - Updates `lastCheckedAt` timestamp

---

## Socket.io Real-time Integration

### Server Setup

**File**: `adminserver/index.js`

```
const io = new SocketIOServer(server, {
  cors: {
    origin: [
      "http://localhost:5173",
      "https://gyapak.in",
      // ... other allowed origins
    ],
    credentials: true,
  },
});

io.on("connection", (socket) => {
  // Client subscribes to specific source
  socket.on("subscribe-source", (sourceCode) => {
    socket.join(sourceCode);
  });
});

global.io = io; // Make available to jobs
```

### Client Connection

**File**: `gyapakAdminPanel/src/Pages/TrackNewUpdates.jsx`

```
const socket = io(API_BASE, {
  withCredentials: true,
  transports: ["websocket"],
});

// Subscribe to all active sources
activeSources.forEach((s) => socket.emit("subscribe-source", s.code));

// Listen for new notifications
socket.on("new-notifications", (incoming) => {
  // Update state with new notifications
  // Deduplicate using seenRef
  // Sort by timestamp
});
```

### Events

| Event | Direction | Payload | Description |
|---|---|---|---|
| `subscribe-source` | Client → Server | `sourceCode: string` | Join room for specific source |
| `new-notifications` | Server → Client | `docs: Notification[]` | New notifications for subscribed source |
| `new-notifications-global` | Server → Client | `docs: Notification[]` | Global broadcast of all new notifications |

---

## Frontend Components

### 1. TrackNewUpdates Component

**File**: `gyapakAdminPanel/src/Pages/TrackNewUpdates.jsx`

**Route**: `/magic-created`

Main dashboard for viewing and managing notifications.

#### Features

**Live Feed**:

- Real-time updates via Socket.io
- Grouped view (by source) or combined view
- Filter by today's notifications
- Search across title, summary, source code, and link

**Statistics**:

- Today's non-result notifications count
- Today's result notifications count
- New notifications since page opened
- Active sources count

**Views**:

- **Grouped**: Notifications organized by source (expandable/collapsible)

- **Combined**: All notifications in chronological order

**Actions**:

- Download today's notifications as Excel
- Open Results Drafts panel
- Open Manage Sources drawer

**State Management**

```
const [sources, setSources] = useState([]);                    // All sources
const [notificationsBySource, setNotificationsBySource] = useState({}); // {
sourceCode: notifications[] }
const [query, setQuery] = useState("");                        // Search query
const [onlyToday, setOnlyToday] = useState(false);            // Filter toggle
const [view, setView] = useState("grouped");                  // View mode
const [expanded, setExpanded] = useState(new Set());          // Expanded sources
```

**Data Flow**

```
sequenceDiagram
    participant User
    participant Component as TrackNewUpdates
    participant API as Backend API
    participant Socket as Socket.io
    participant State as React State

    User->>Component: Opens /magic-created
    Component->>API: GET /api/sources
    API-->>Component: sources[]
    Component->>State: setSources(sources)

    Component->>API: GET /api/notifications (for each source)
    API-->>Component: notifications[]
    Component->>State: setNotificationsBySource({...})

    Component->>Socket: Connect & subscribe to sources

    loop Real-time updates
        Socket-->>Component: new-notifications event
        Component->>Component: Deduplicate using seenRef
        Component->>State: Update notificationsBySource
        Component->>State: Increment newSinceOpenCount
    end

    User->>Component: Search/Filter
    Component->>Component: Filter in memory
    Component->>User: Display filtered results
```

## 2. ManageSources Component

**File**: `gyapakAdminPanel/src/Pages/ManageSources.jsx`

CRUD interface for managing sources.

#### Features

- **Create**: Add new sources with validation
- **Read**: View all sources with pagination and search
- **Update**: Edit source configuration
- **Delete**: Remove sources (with confirmation)
- **Toggle Active**: Enable/disable scraping for a source

#### Form Fields

| Field | Type | Required | Description |
|---|---|---|---|
| Code | Text | Yes | Unique identifier |
| Name | Text | Yes | Display name |
| Base URL | URL | Yes | Base URL for link resolution |
| Notification URL | URL | Yes | URL to scrape |
| Type | Select | Yes | html / rss / json |
| Selector | Text | No | CSS selector (for HTML) |
| Interval | Number | Yes | Minutes between scrapes |

### 3. ResultNotificationsPanel Component

**File**: `gyapakAdminPanel/src/Pages/ResultNotificationsPanel.jsx`

Specialized panel for handling result-related notifications.

#### Features

**Filtering**:

- Automatically identifies result notifications using keyword matching
- Filters notifications containing "result" in type, category, title, or summary

**Draft Management**:

- Converts notifications to draft payloads
- View drafts as JSON
- Copy to clipboard
- Download as JSON file

**Batch Editing**:

- Select multiple notifications
- Edit fields: title, link, resultDate, isTentative, description
- Validate before upload
- Bulk upload to `/api/results/import`

**Draft Payload Structure**:

```
{
  kind: "result",
  title: "Notification Title",
  link: "https://example.com/result",
  description: "",
  resultDate: "2024-01-15", // YYYY-MM-DD or null
  isTentative: false
}
```

---

## Complete Data Flow

### 1. Source Creation Flow

```
sequenceDiagram
    participant Admin
    participant UI as ManageSources UI
    participant API as Source Controller
    participant DB as MongoDB

    Admin->>UI: Fill form & submit
    UI->>API: POST /api/sources
    API->>API: Validate required fields
    API->>DB: Check if code exists
    alt Code already exists
        DB-->>API: Found
        API-->>UI: 409 Conflict
        UI-->>Admin: Error message
    else Code is unique
        DB-->>API: Not found
        API->>DB: Create source document
        DB-->>API: Created source
        API-->>UI: 201 Created
        UI->>UI: Refresh sources list
        UI-->>Admin: Success message
    end
```

### 2. Automated Scraping Flow

```
sequenceDiagram
    participant Cron
    participant Job as Monitor Job
    participant Scraper
    participant Website as External Website
    participant NotifService
    participant DB as MongoDB
    participant Socket as Socket.io
    participant Client as Admin Panel
    Cron->>Job: Every 2 minutes
    Job->>DB: Find active sources
    DB-->>Job: sources[]

    loop For each source
        Job->>Scraper: fetchItemsForSource(source)

        alt Type is RSS
            Scraper->>Website: GET RSS feed
            Website-->>Scraper: XML response
            Scraper->>Scraper: Parse with rss-parser
        else Type is HTML
            Scraper->>Website: GET HTML page
            Website-->>Scraper: HTML response
            Scraper->>Scraper: Parse with Cheerio
            Scraper->>Scraper: Extract links using selector
        end

        Scraper-->>Job: items[]

        Job->>NotifService: saveNewNotifications(source, items)

        loop For each item
            NotifService->>NotifService: Create hash (SHA-256)
            NotifService->>DB: Upsert notification
        end

        DB-->>NotifService: New documents
        NotifService-->>Job: newDocs[]

        alt Has new notifications
            Job->>Job: Filter last 4 days
            Job->>Socket: Emit to room(sourceCode)
            Socket-->>Client: new-notifications event
            Client->>Client: Update UI in real-time
        end

        Job->>DB: Update source.lastCheckedAt
    end
```

### 3. Real-time Notification Flow

```
sequenceDiagram
    participant Job as Monitor Job
    participant Socket as Socket.io Server
    participant Client as TrackNewUpdates
    participant State as React State
    participant UI as User Interface

    Job->>Socket: emit("new-notifications", freshDocs)
    Socket->>Client: Broadcast to subscribed clients
    Client->>Client: Check if already seen (seenRef)

    alt Not seen before
        Client->>State: Add to notificationsBySource
        Client->>State: Add to newSinceOpenRef
        Client->>State: Increment newSinceOpenCount
        Client->>Client: Sort by timestamp
        Client->>UI: Re-render with new notification
        UI->>UI: Show blue dot indicator
    else Already seen
        Client->>Client: Skip (deduplicate)
    end
```

### 4. Result Upload Flow

```
sequenceDiagram
    participant Admin
    participant Panel as ResultNotificationsPanel
    participant EditModal as Edit Modal
    participant API as Backend API
    participant DB as MongoDB

    Admin->>Panel: Select notifications
    Panel->>Panel: Filter result notifications
    Admin->>Panel: Click "Edit & Upload"
    Panel->>EditModal: Open with selected drafts

    Admin->>EditModal: Edit fields (title, link, date, etc.)
    EditModal->>EditModal: Update local state

    Admin->>EditModal: Click "Final Upload"
    EditModal->>EditModal: Validate drafts

    alt Validation fails
        EditModal-->>Admin: Show error message
    else Validation passes
        EditModal->>API: POST /api/results/import
        API->>DB: Bulk insert results
        DB-->>API: Success
        API-->>EditModal: 200 OK
        EditModal->>Panel: Close modals
        Panel->>Panel: Clear selection
        EditModal-->>Admin: Success alert
    end
```

---

## API Reference

### Sources API

**GET /api/sources**

Fetch all sources.

**Response**:

```json
{
  "success": true,
  "data": [
    {
      "_id": "507f1f77bcf86cd799439011",
      "code": "CENTRAL_UPSC",
      "name": "UPSC",
      "baseUrl": "https://www.upsc.gov.in",
      "notificationUrl": "https://www.upsc.gov.in/recruitment/recruitment-
advertisement",
      "type": "html",
      "selector": ".notice-list a",
      "intervalMinutes": 5,
      "lastCheckedAt": "2024-01-15T10:30:00.000Z",
      "isActive": true,
      "createdAt": "2024-01-01T00:00:00.000Z",
      "updatedAt": "2024-01-15T10:30:00.000Z"
    }
  ]
}
```

**POST /api/sources**

Create a new source.

**Request Body**:

```json
{
  "code": "CENTRAL_SSC",
  "name": "SSC",
  "baseUrl": "https://ssc.nic.in",
  "notificationUrl": "https://ssc.nic.in/notifications",
  "type": "html",
  "selector": ".notification-list a",
  "intervalMinutes": 10
}
```

**Response**: `201 Created`

**PUT /api/sources/:id**

Update an existing source.

**Request Body**: Same as POST (all fields optional)

**Response**: `200 OK`

**DELETE /api/sources/:id**

Delete a source.

**Response**: `200 OK`

### Notifications API

**GET /api/notifications**

Fetch notifications with optional filtering.

**Query Parameters**:

- `sourceCode` (optional): Filter by source code
- `limit` (optional): Number of results (default: 50)

**Response**:

```json
{
  "success": true,
  "data": [
    {
      "_id": "507f1f77bcf86cd799439012",
      "sourceId": "507f1f77bcf86cd799439011",
      "sourceCode": "CENTRAL_UPSC",
      "title": "Combined Defence Services Examination (I), 2024",
      "link": "https://www.upsc.gov.in/examinations/cds-i-2024",
      "summary": "Applications are invited for CDS Exam",
      "rawText": null,
      "publishedAt": "2024-01-15T00:00:00.000Z",
      "firstSeenAt": "2024-01-15T10:30:00.000Z",
      "itemHash": "abc123...",
      "createdAt": "2024-01-15T10:30:00.000Z",
      "updatedAt": "2024-01-15T10:30:00.000Z"
    }
  ]
}
```

---

## Key Features Explained

### 1. Deduplication Strategy

The system uses multiple layers of deduplication:

**Hash-based**:

- SHA-256 hash created from `sourceCode|title|link`
- Stored in `itemHash` field
- Compound unique index on `{ sourceCode, itemHash }`

**Link-based**:

- Compound unique index on `{ sourceCode, link }`

- Prevents duplicate URLs from same source

**Client-side**:

- `seenRef` tracks notification keys in memory
- Prevents duplicate rendering in real-time updates

### 2. Time Window Filtering

**4-Day Window**:

- Only notifications from last 4 days are broadcast via Socket.io
- Reduces noise from historical backfill
- Configurable in `monitorSources.job.js`

**IST Timezone**:

- Frontend uses IST for "today" calculations
- `istDateKey` function converts dates to IST date strings
- Ensures consistent "today" filtering across timezones

### 3. Result Detection

**Keyword Matching**:

```
const isResultNotification = (n) => {
  const hay = `${n?.type || ""} ${n?.category || ""} ${n?.title || ""} ${n?.summary
|| ""}`.toLowerCase();
  return hay.includes("result");
};
```

Automatically separates result notifications for specialized handling.

### 4. Pagination & Performance

**Sources List**:

- Client-side pagination (10/20/30 per page)
- Search filtering before pagination
- Maintains page state across filters

**Notifications**:

- Limited to 60 per source on initial load
- Socket.io updates prepend to existing list
- Maximum 400 notifications cached per source
- Display limit of 30 in result panel (with note)

---

## Configuration

### Cron Schedule

**File**: `adminserver/index.js`

```
// Monitor sources every 2 minutes
cron.schedule("*/2 * * * *", async () => {
  await monitorAllSources(io);
});
```

### Socket.io CORS

Allowed origins configured in `adminserver/index.js` :

```
cors: {
  origin: [
    "http://localhost:5173",
    "http://localhost:5174",
    "https://gyapak.in",
    "https://www.gyapak.in",
    // ... more origins
  ],
  credentials: true,
}
```

### Request Timeout

Default: 60 seconds (configurable per source via `requestTimeoutMs` )

---

## Error Handling

### Scraper Service

- Catches HTTP errors and logs them
- Returns empty array on failure
- Continues with next source on error
- Handles malformed RSS feeds gracefully

### Socket.io

- Automatic reconnection on disconnect
- Graceful degradation (UI still works without real-time)
- Client-side deduplication prevents duplicate processing

### Database

- Unique indexes prevent duplicate inserts
- `bulkWrite` with `ordered: false` continues on errors
- Upsert strategy ensures idempotency

---

## Monitoring & Debugging

### Logs

**Scraper Activity**:

```
RSS fetch for CENTRAL_UPSC https://...
RSS items fetched for CENTRAL_UPSC: 25
Fresh notifications for CENTRAL_UPSC in last 4 days: 3
```

**Socket.io**:

```
Client connected: socket-id-123
Socket socket-id-123 joined CENTRAL_UPSC
Client disconnected: socket-id-123
```

### Database Queries

**Find active sources**:

```
db.sources.find({ isActive: true })
```

**Recent notifications**:

```
db.notifications.find({ sourceCode: "CENTRAL_UPSC" })
  .sort({ firstSeenAt: -1 })
  .limit(10)
```

**Duplicates check**:

```
db.notifications.aggregate([
  { $group: { _id: "$itemHash", count: { $sum: 1 } } },
  { $match: { count: { $gt: 1 } } }
])
```

---

## Best Practices

### Adding New Sources

1. **Test URL first**: Verify the notification URL loads correctly
2. **Inspect HTML structure**: Use browser DevTools to find correct CSS selector
3. **Start with longer interval**: Use 10-15 minutes initially, reduce if needed
4. **Monitor logs**: Check scraper logs for errors after creation
5. **Verify deduplication**: Ensure notifications aren't duplicated

### Managing Notifications

1. **Regular cleanup**: Archive old notifications (>30 days)
2. **Monitor database size**: Notifications can grow quickly
3. **Review failed scrapes**: Check logs for sources with 0 items
4. **Update selectors**: Websites change structure; update selectors accordingly

### Performance Optimization

1. **Limit notification cache**: Keep max 400 per source in memory
2. **Use pagination**: Don't load all notifications at once
3. **Optimize selectors**: Specific selectors are faster than broad ones

4. **Monitor socket connections**: Limit concurrent connections

---

## Troubleshooting

### No notifications appearing

**Check**:

1. Is source active? ( `isActive: true` )
2. Has enough time passed? (check `intervalMinutes` )
3. Are there scraper errors in logs?
4. Is CSS selector correct?
5. Is website blocking requests? (check User-Agent)

### Duplicate notifications

**Check**:

1. Are unique indexes present?
2. Is `itemHash` being generated correctly?
3. Are multiple sources scraping same URL?

### Socket.io not connecting

**Check**:

1. Is origin allowed in CORS config?
2. Is WebSocket transport enabled?
3. Are credentials being sent?
4. Check browser console for errors

### Slow performance

**Check**:

1. Database indexes present?
2. Too many notifications cached?
3. Pagination working correctly?
4. Network latency to external sites?

---

## Future Enhancements

### Potential Improvements

1. **AI-powered categorization**: Use LLM to auto-categorize notifications
2. **Webhook support**: Allow external systems to push notifications
3. **Advanced filtering**: Tags, categories, priority levels
4. **Notification history**: Track changes to notifications over time
5. **Analytics dashboard**: Scraping success rates, response times
6. **Retry mechanism**: Automatic retry for failed scrapes
7. **Rate limiting**: Respect website rate limits
8. **Proxy support**: Rotate proxies for scraping
9. **Screenshot capture**: Save visual proof of notifications
10. **Email alerts**: Notify admins of critical updates

---

## Summary

The Magic Tool is a comprehensive notification tracking system that:

1. **Automates** web scraping from multiple sources (HTML/RSS)
2. **Deduplicates** notifications using hash-based and index-based strategies
3. **Delivers** real-time updates via Socket.io
4. **Filters** result notifications for specialized processing
5. **Enables** batch editing and uploading of notifications
6. **Provides** intuitive UI for source and notification management

The system is designed for scalability, reliability, and ease of use, making it an essential tool for monitoring government job notifications and announcements.