Tasks & Events Management System

A discord bot with an accompanying web application for personal use in managing events, appointments, tasks, and recurring items with reminders sent to a private Discord server.

Created by Destiny Kaplan

Table of Contents

Overview

Purpose

This project was created in an effort to streamline and increase communication about new tasks and events for users utilizing a personal, private family discord server.

Scope

This project allows for scheduling of tasks and events through the accompanying Flask web application and the Discord bot commands, and serves to supply the users with notifications of creation, due dates, and completion of tasks and events through both Discord bot commands for remote ease-of-use (such as mobile) as well as a web application interface for computers.

The project is local-hosted, supplying those on LAN to also access the web application with ease, and allows for personal completion analytics using an analytics dashboard within the web application.

Functions

Core Functionality

Task & Event Management

- **Dual Item Types**: Separate handling for tasks and events with distinct properties
- **Status Tracking**: Pending/completed status management with completion notifications
- **Priority System**: Three-tier priority levels (Low, Medium, High) with visual indicators
- **Category Organization**: Predefined categories (Home, Vehicle, Bills, School, Work, Misc)
- **Notes & Descriptions**: Rich text support for additional item details
- **Custom Colors**: User-selectable colors for visual organization in calendar view

Date & Time Management

- **Flexible Date Input**: Multiple format support (12-hour, 24-hour, date-only)
- **Recurring Items**: Daily, weekly, and monthly repeat options
- **Due Date Tracking**: Automatic overdue and due-soon detection
- **Timeline Views**: Multiple calendar perspectives (monthly, weekly, daily, list)

Discord Bot Integration

Command Interface

- **Channel Restrictions**: Commands limited to designated channels for server organization
- **Rich Embeds**: Formatted responses with color-coded information
- **Interactive Commands**: 12 distinct commands for complete system control
- **Search Functionality**: Text-based searching across all item fields
- **Batch Operations**: List views showing multiple items with IDs for easy reference

Automated Notifications

- **Reminder System**: 30-minute advance warnings before due dates
- **Daily Summaries**: Comprehensive morning reports at 8:00 AM
- **Real-time Updates**: Instant notifications for item creation, completion, and changes
- **Smart Mentions**: User tagging support with spam protection
- **Status Broadcasting**: Automatic announcements for important changes

User Experience

- **Error Handling**: Graceful error messages with usage guidance
- **Input Validation**: Comprehensive data validation with helpful feedback
- **Debug Tools**: System status and diagnostic commands
- **Permission Management**: Role-based access control for server administration

Web Application Interface

Dashboard Features

- **Unified Overview**: Combined task and event display with priority indicators
- **Mini Calendar**: Embedded calendar widget showing upcoming items
- **Quick Actions**: One-click completion and status updates
- **Time Indicators**: Visual alerts for overdue and due-soon items
- **Responsive Design**: Mobile-friendly interface with adaptive layouts

Full Calendar Integration

- **FullCalendar.js Integration**: Professional calendar interface with multiple view modes
- **Drag & Drop**: Visual rescheduling by dragging events
- **Click to Edit**: Inline editing with modal dialogs
- **Recurring Display**: Automatic generation and display of recurring instances
- **Color Coding**: Custom colors for visual organization
- **Event Details**: Comprehensive popup information panels

Advanced Analytics

- **Statistics Dashboard**: Real-time metrics and completion rates
- **Visual Charts**: Interactive charts using Chart.js library
- **Trend Analysis**: Historical data visualization and patterns
- **Filter Options**: Time-based and type-based data filtering
- **Export Capabilities**: Data visualization for reporting purposes

Technical Architecture

Database Management

- **SQLite Backend**: Lightweight, serverless database with automatic setup
- **Migration System**: Automatic schema updates and column additions
- **Data Integrity**: Foreign key constraints and validation rules
- **Backup Friendly**: Single-file database for easy backup and portability
- **Performance Optimization**: Indexed queries and efficient data retrieval

Security Framework

- **Authentication System**: Secure user login with hashed passwords
- **Session Management**: Flask-based session handling with configurable security
- **Input Sanitization**: Comprehensive XSS and injection protection
- **Rate Limiting**: Built-in protection against abuse and spam
- **Environment Security**: Sensitive data stored in environment variables

Automation & Scheduling

- **APScheduler Integration**: Robust job scheduling for reminders and updates
- **Thread-Safe Operations**: Concurrent handling of web and Discord operations
- **Error Recovery**: Automatic retry mechanisms and graceful degradation
- **Resource Management**: Efficient memory and connection pooling

Advanced Features

Multi-Platform Synchronization

- **Real-Time Sync**: Changes instantly reflected across Discord and web interfaces
- **Conflict Resolution**: Automatic handling of simultaneous edits
- **Event Propagation**: Notifications sent across all active channels

Recurring Event Intelligence

- **Pattern Recognition**: Smart generation of recurring instances
- **Date Range Optimization**: Efficient handling of long-term recurring patterns
- **Exception Handling**: Graceful management of scheduling conflicts
- **Calendar Integration**: Seamless display of recurring patterns

Notification Intelligence

- **Smart Timing**: Context-aware notification scheduling
- **Mention Management**: Automatic user tagging with spam prevention
- **Message Formatting**: Rich formatting with emojis and structured content
- **Delivery Confirmation**: Error handling for failed notifications

Data Management

- **Cleanup Automation**: Automatic removal of old completed items
- **Data Validation**: Comprehensive input checking and sanitization
- **Export Options**: Multiple data formats for external use
- **Import Capabilities**: Bulk data entry and migration support

User Interface Features

Accessibility

- **Keyboard Navigation**: Full keyboard support for all functions
- **Screen Reader Compatible**: Semantic HTML and ARIA labels
- **Color Contrast**: High contrast design for visual accessibility
- **Responsive Layout**: Mobile-first design with touch-friendly controls

Customization Options

- **Color Themes**: User-selectable colors for personal organization
- **View Preferences**: Multiple calendar and list view options
- **Notification Settings**: Customizable alert timing and channels
- **Category Management**: Flexible categorization system

Performance Features

- **Lazy Loading**: Efficient data loading for large datasets
- **Caching System**: Intelligent caching of frequently accessed data
- **Optimized Queries**: Database optimization for fast response times
- **Auto-Refresh**: Periodic updates without manual intervention

Integration Capabilities

Discord Server Integration

- **Multi-Channel Support**: Separate command and notification channels
- **Permission Integration**: Discord role-based access control
- **Server Customization**: Configurable behavior per Discord server
- **Bot Management**: Easy setup and configuration tools

External System Compatibility

- **Calendar Export**: Standard calendar format compatibility
- **API Endpoints**: RESTful API for third-party integrations
- **Webhook Support**: External notification capabilities
- **Data Portability**: Standard formats for data exchange

Development & Maintenance

- **Modular Architecture**: Clean separation of concerns for easy maintenance
- **Configuration Management**: Environment-based configuration system

- **Logging System**: Comprehensive error and activity logging
- **Update Mechanism**: Built-in system for feature updates and bug fixes

Automated Features

Reminder System

- **30-Minute Warnings**: Automatic reminders before due dates
- **Daily Updates**: Morning summary at 8:00 AM with:
  - Overdue items (red alerts)
  - Due today items
  - Upcoming items (next 7 days)
  - Statistics summary

Recurring Items

- **Repeat Intervals**: None, Daily, Weekly, Monthly
- **Automatic Generation**: Future instances created automatically
- **Calendar Integration**: Shows recurring patterns

Discord Notifications

- **New Item Alerts**: When items are created via web
- **Completion Notifications**: When items are marked complete
- **Daily Summaries**: Comprehensive morning reports
- **Reminder Messages**: Pre-due date notifications

Security Features

Authentication & Authorization

- **User Management**: Secure user authentication with hashed passwords using Werkzeug's password hashing
- **Session Management**: Flask sessions with configurable secret keys
- **Login Protection**: All web routes protected with login requirements
- **Rate Limiting**: Built-in protection against brute force attacks

Database Security

- **Input Validation**: Comprehensive validation for all user inputs with length limits
- **SQL Injection Protection**: Parameterized queries throughout the application
- **Secure File Permissions**: Database files automatically set to owner-only access (600)
- **Data Sanitization**: All user inputs are sanitized and truncated to prevent injection attacks

Discord Bot Security

- **Channel Restrictions**: Bot commands restricted to designated channels only
- **Input Validation**: All Discord commands validate and sanitize user input
- **Mention Controls**: @everyone mentions filtered out to prevent spam
- **Error Handling**: Comprehensive error handling prevents information disclosure

Environment Security

- **Environment Variables**: Sensitive data stored in .env files (not in code)
- **Debug Mode**: Debug mode disabled by default for production
- **Secure Defaults**: Conservative security settings throughout

Installation Guide

Prerequisites

- Python 3.7 or higher
- Discord Bot Token
- Discord Server with appropriate permissions

Step 1: Download and Extract Files

1. Download all the provided files
2. Create a new directory for your bot: `mkdir tasks-events-bot`
3. Place all files in the directory

Step 2: Install Dependencies

```
cd tasks-events-bot

pip install -r requirements.txt
```

Step 3: Create Environment Configuration

Create a `.env` file in the root directory with the following content:

```
# Discord Bot Configuration
DISCORD_TOKEN=your_discord_bot_token_here
DISCORD_COMMANDS_CHANNEL_ID=your_commands_channel_id
DISCORD_NOTIFICATIONS_CHANNEL_ID=your_notifications_channel_id

# Flask Configuration
SECRET_KEY=your_secret_key_here_make_it_long_and_random
DEBUG=False
```

Step 4: Get Discord Bot Token

1. Go to https://discord.com/developers/applications
2. Click "New Application" and give it a name
3. Go to "Bot" section and click "Add Bot"
4. Copy the bot token and paste it in your `.env` file
5. Enable "Message Content Intent" under "Privileged Gateway Intents"

Step 5: Get Discord Channel IDs

1. Enable Developer Mode in Discord (User Settings > Advanced > Developer Mode)
2. Right-click on your commands channel and select "Copy ID"
3. Right-click on your notifications channel and select "Copy ID"
4. Paste these IDs in your `.env` file

Step 6: Invite Bot to Server

1. In Discord Developer Portal, go to OAuth2 > URL Generator
2. Select scopes: `bot` and `applications.commands`
3. Select bot permissions: `Send Messages`, `Read Message History`, `Use Slash Commands`
4. Use the generated URL to invite the bot to your server

Step 7: Initialize Database and Create User

```
python setup.py
```

Follow the prompts to create your first user account.

Step 8: Update User Mentions

1. In the main.py file, go to line 757 of the code just after the Scheduler Setup to update the specific users that should be allowed for mention pings.

```
    # Predefine users
    USER_ID_MAP = {
    "UserNumberOne": 123456789012345678,
    "UserNumberTwo": 23456789012345678,
  }
```

Step 9: Start the Application

```
python main.py
```

The bot will start and display:

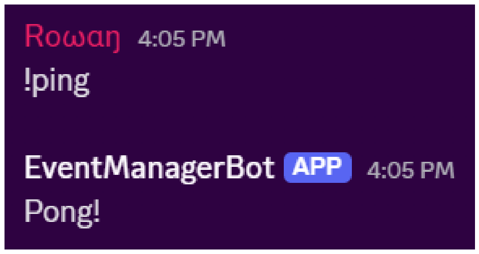
- Web application URL (typically http://127.0.0.1:5000)
- Discord bot connection status
- Scheduler initialization

Discord Bot Commands

Basic Commands

!ping

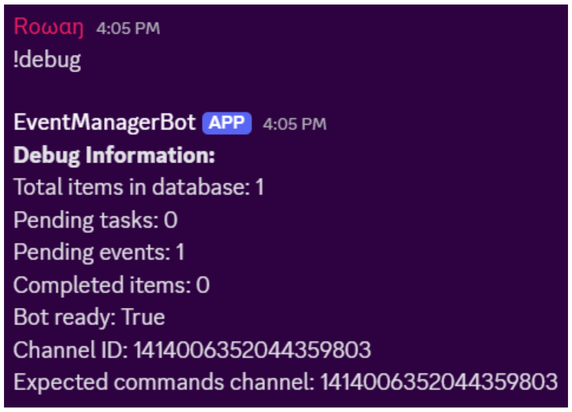
- **Usage**: `!ping`
- **Description**: Test if the bot is responsive
- **Example**: `!ping` → Bot responds with "Pong!"



```
Rowan 4:05 PM
!ping

EventManagerBot APP 4:05 PM
Pong!
```

!debug

- **Usage**: `!debug`
- **Description**: Shows system status and statistics
- **Output**: Total items, pending tasks/events, bot status



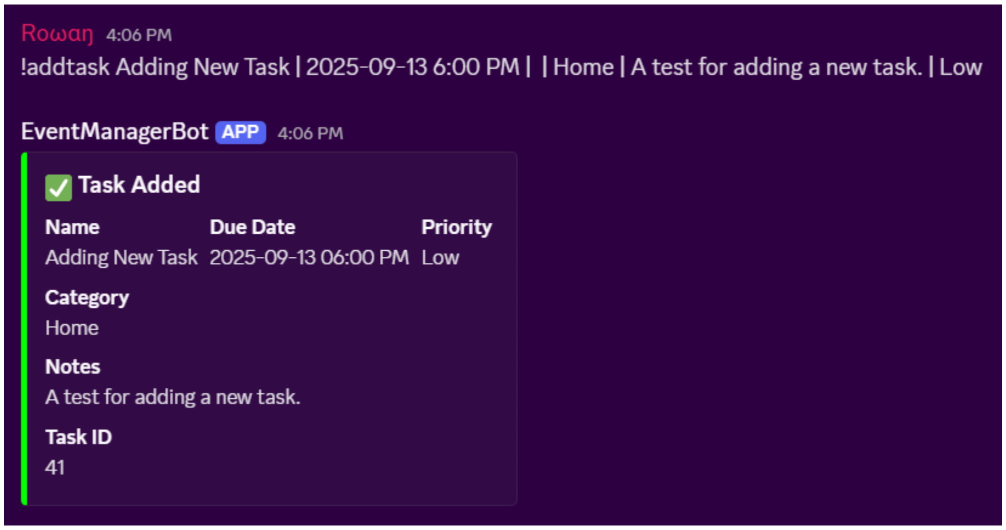
```
Rowan 4:05 PM
!debug

EventManagerBot APP 4:05 PM
Debug Information:
Total items in database: 1
Pending tasks: 0
Pending events: 1
Completed items: 0
Bot ready: True
Channel ID: 1414006352044359803
Expected commands channel: 1414006352044359803
```

Task Management

!addtask

- **Usage**: `!addtask <name> | <due_date> | [mention] | [category] | [notes] | [priority]`
- **Description**: Creates a new task

- **Required**: name, due_date
- **Optional**: mention, category, notes, priority
- **Date Formats**:
  - `YYYY-MM-DD HH:MM` (24-hour)
  - `YYYY-MM-DD H:MM AM/PM` (12-hour)
  - `YYYY-MM-DD` (date only)
- **Examples**:
  - !addtask Homework | 2024-09-15 23:59 | @john | School | Chapter 5 problems | High
  - !addtask Meeting prep | 2024-09-16 2:00 PM | | Work | | Medium
  - !addtask Buy groceries | 2024-09-14



Rowan 4:06 PM
!addtask Adding New Task | 2025-09-13 6:00 PM | | Home | A test for adding a new task. | Low

EventManagerBot APP 4:06 PM

✅ Task Added

Name	Due Date	Priority
Adding New Task	2025-09-13 06:00 PM	Low

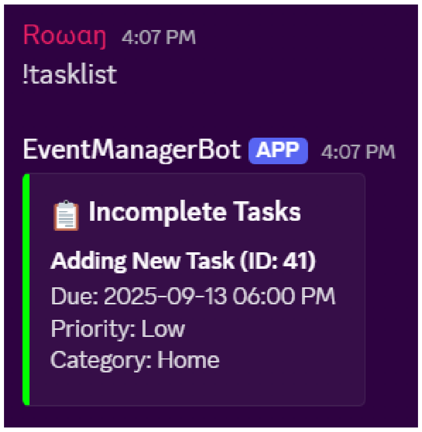
Category
Home

Notes
A test for adding a new task.

Task ID
41

!tasklist

- **Usage**: `!tasklist`
- **Description**: Shows all incomplete tasks
- **Output**: List of pending tasks with IDs, due dates, priorities



Rowan 4:07 PM
!tasklist

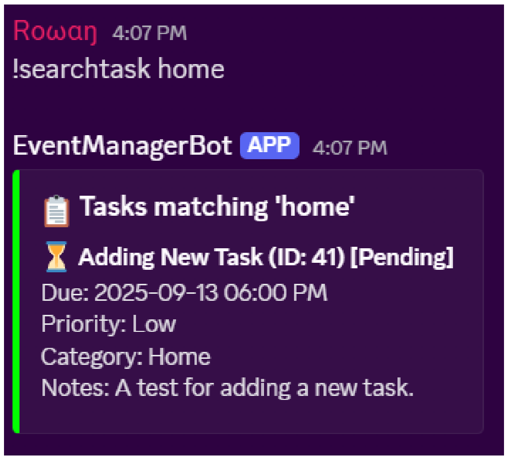
EventManagerBot APP 4:07 PM

📋 Incomplete Tasks

Adding New Task (ID: 41)
Due: 2025-09-13 06:00 PM
Priority: Low
Category: Home

!searchtask

- **Usage**: `!searchtask <search_term>`
- **Description**: Search tasks by name, category, or notes
- **Example**: `!searchtask homework` → Shows all tasks containing "homework"

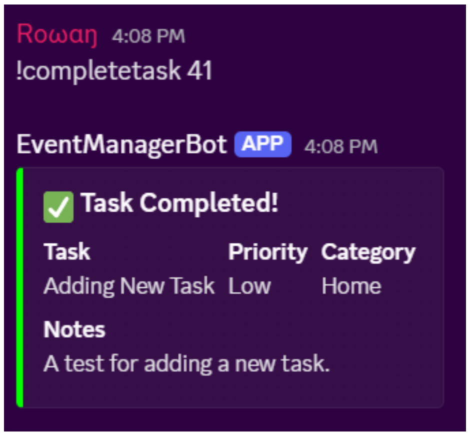


Rowan 4:07 PM
!searchtask home

EventManagerBot APP 4:07 PM
📋 Tasks matching 'home'
⏳ Adding New Task (ID: 41) [Pending]
Due: 2025-09-13 06:00 PM
Priority: Low
Category: Home
Notes: A test for adding a new task.

!completetask

- **Usage**: `!completetask <task_id>`
- **Description**: Marks a task as completed
- **Example**: `!completetask 123`



Rowan 4:08 PM
!completetask 41

EventManagerBot APP 4:08 PM
✅ Task Completed!

Task	Priority	Category
Adding New Task	Low	Home

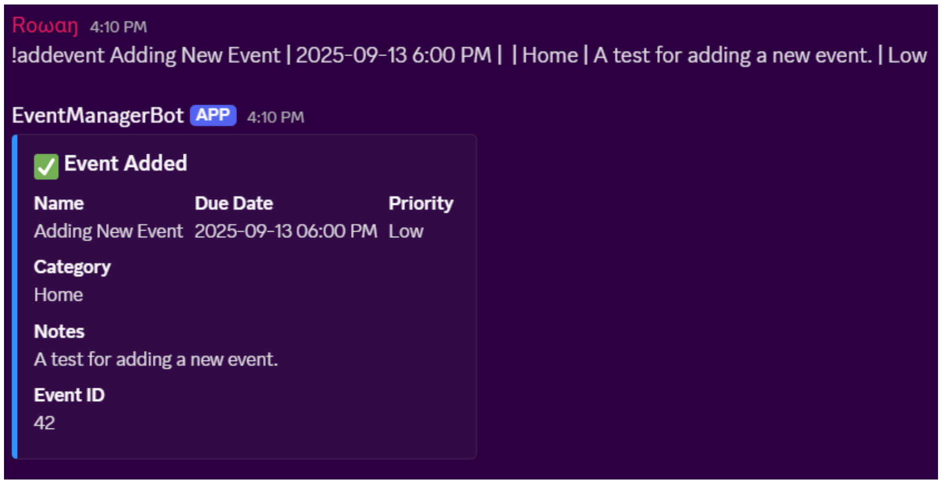
Notes
A test for adding a new task.

Event Management

!addevent

- **Usage**: `!addevent <name> | <due_date> | [mention] | [category] | [notes] | [priority]`

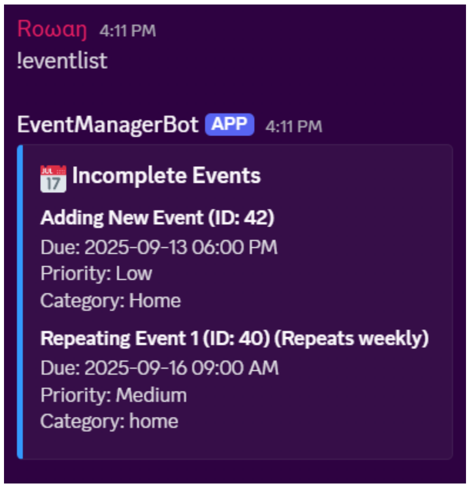
- **Description**: Creates a new event
- **Same parameters as !addtask**
- **Examples**:
  - !addevent Team Meeting | 2024-09-16 10:00 AM | @team | Work | Weekly standup | High
  - !addevent Birthday Party | 2024-09-20 6:00 PM | | Personal | Bring cake



```
Rowan 4:10 PM
!addevent Adding New Event | 2025-09-13 6:00 PM | | Home | A test for adding a new event. | Low

EventManagerBot APP 4:10 PM
✅ Event Added
Name              Due Date             Priority
Adding New Event  2025-09-13 06:00 PM  Low
Category
Home
Notes
A test for adding a new event.
Event ID
42
```

!eventlist

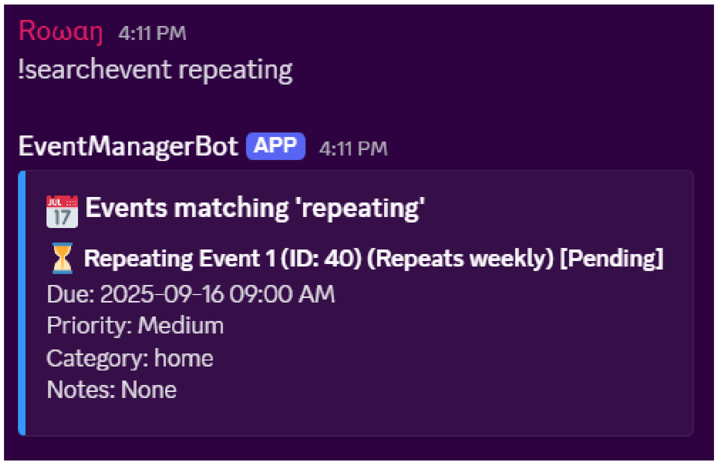
- **Usage**: `!eventlist`
- **Description**: Shows all incomplete events



```
Rowan 4:11 PM
!eventlist

EventManagerBot APP 4:11 PM
📅 Incomplete Events
Adding New Event (ID: 42)
Due: 2025-09-13 06:00 PM
Priority: Low
Category: Home
Repeating Event 1 (ID: 40) (Repeats weekly)
Due: 2025-09-16 09:00 AM
Priority: Medium
Category: home
```

!searchevent

- **Usage**: `!searchevent <search_term>`
- **Description**: Search events by name, category, or notes



Rowan 4:11 PM
!searchevent repeating

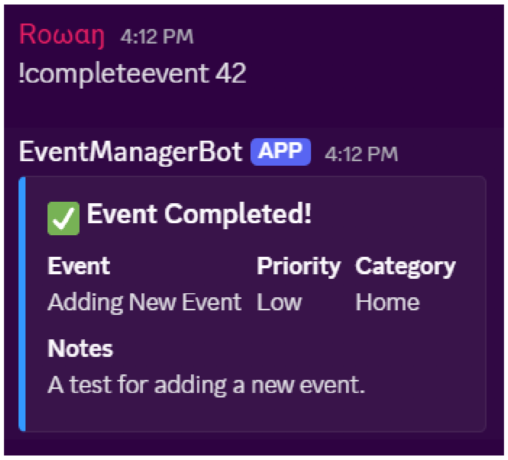
EventManagerBot APP 4:11 PM

📅 **Events matching 'repeating'**

⏳ **Repeating Event 1 (ID: 40) (Repeats weekly) [Pending]**
Due: 2025-09-16 09:00 AM
Priority: Medium
Category: home
Notes: None

!completeevent

- **Usage**: `!completeevent <event_id>`
- **Description**: Marks an event as completed



Rowan 4:12 PM
!completeevent 42

EventManagerBot APP 4:12 PM

✅ **Event Completed!**

Event	**Priority**	**Category**
Adding New Event	Low	Home

Notes
A test for adding a new event.

Parameters Guide

Categories: Home, Vehicle, Bills, School, Work, Misc

Priorities: Low, Medium, High

Date Formats:

- `2024-09-15 14:30` (2:30 PM)
- `2024-09-15 2:30 PM`
- `2024-09-15` (defaults to start of day)

Mentions:

- Use Discord usernames without @ symbol
- Bot will format them automatically
- @everyone is not allowed

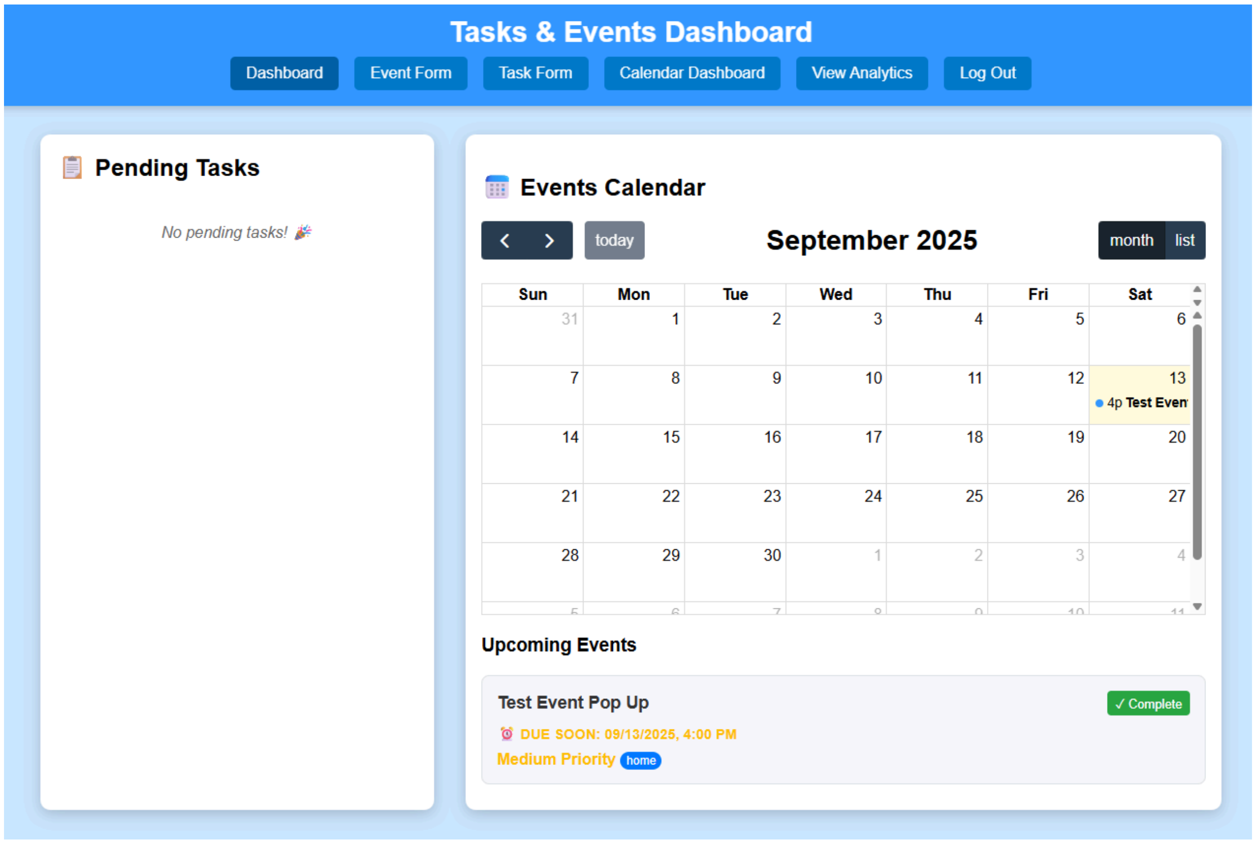
Web Application Usage

Accessing the Web Interface

1. Start the bot: `python main.py`
2. Open browser to http://127.0.0.1:5000
   a. Other LAN users can utilize the host IP and port 5000 to access the web app if necessary.
3. Login with credentials created during setup

Dashboard Features

- **Task Overview**: All pending tasks with priority indicators
- **Event Calendar**: Mini calendar showing upcoming events
- **Quick Actions**: Complete tasks/events directly from dashboard
- **Status Indicators**: Color-coded overdue/due soon alerts



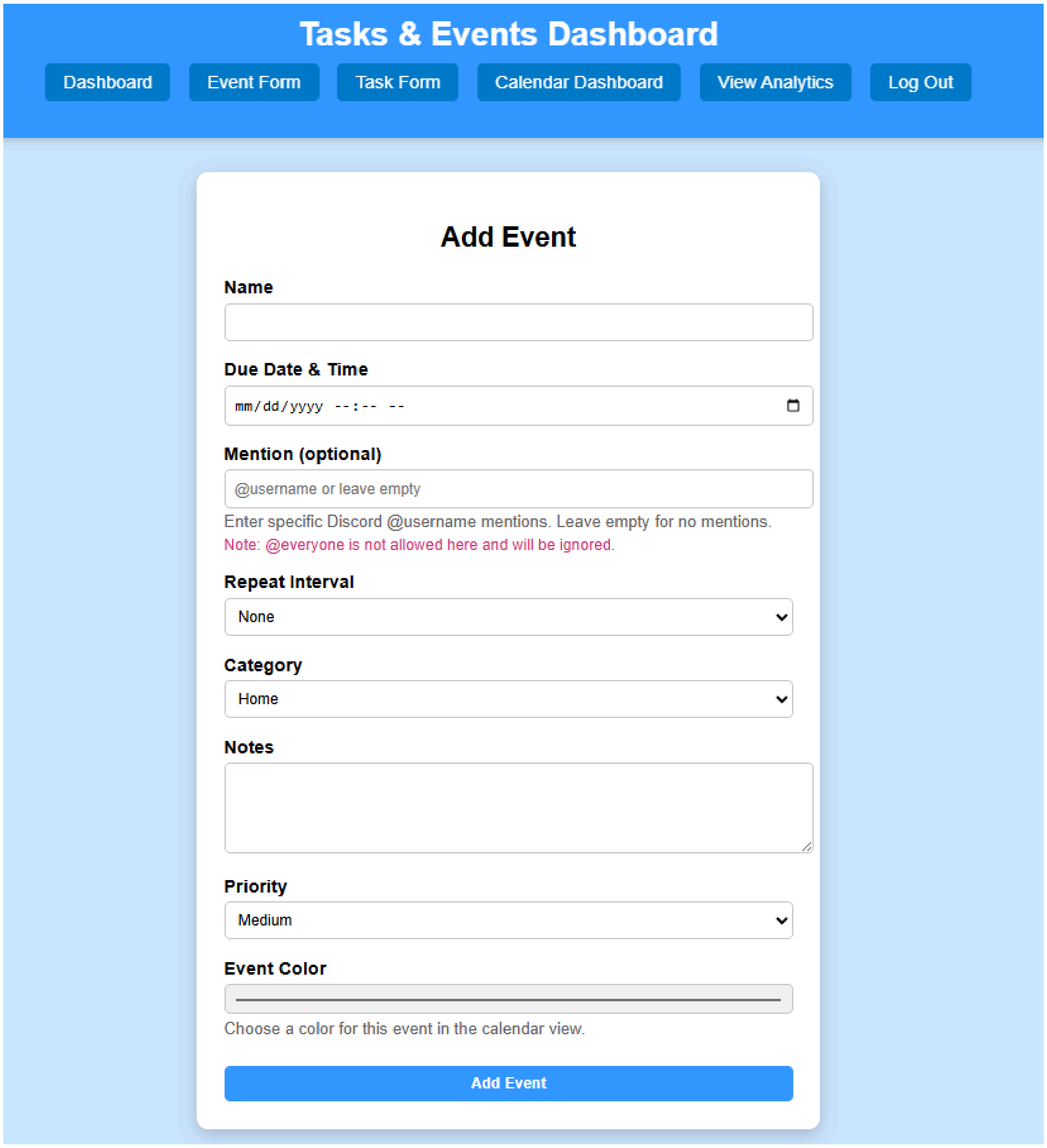
Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	1	2	3	4	5	6
7	8	9	10	11	12	13 ● 4p Test Even
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4

Test Event Pop Up
⏰ DUE SOON: 09/13/2025, 4:00 PM
Medium Priority home

Adding Items via Web

1. Click "Event Form" or "Task Form" in navigation

2. Fill out the form:
   - **Name**: Required title
   - **Due Date**: Use date/time picker
   - **Mention**: Optional Discord username
   - **Category**: Dropdown selection
   - **Priority**: Low/Medium/High
   - **Notes**: Optional description
   - **Color**: Visual identifier for calendar



Add Event

Name

Due Date & Time
mm/dd/yyyy --:-- --

Mention (optional)
@username or leave empty
Enter specific Discord @username mentions. Leave empty for no mentions.
Note: @everyone is not allowed here and will be ignored.

Repeat Interval
None

Category
Home

Notes

Priority
Medium

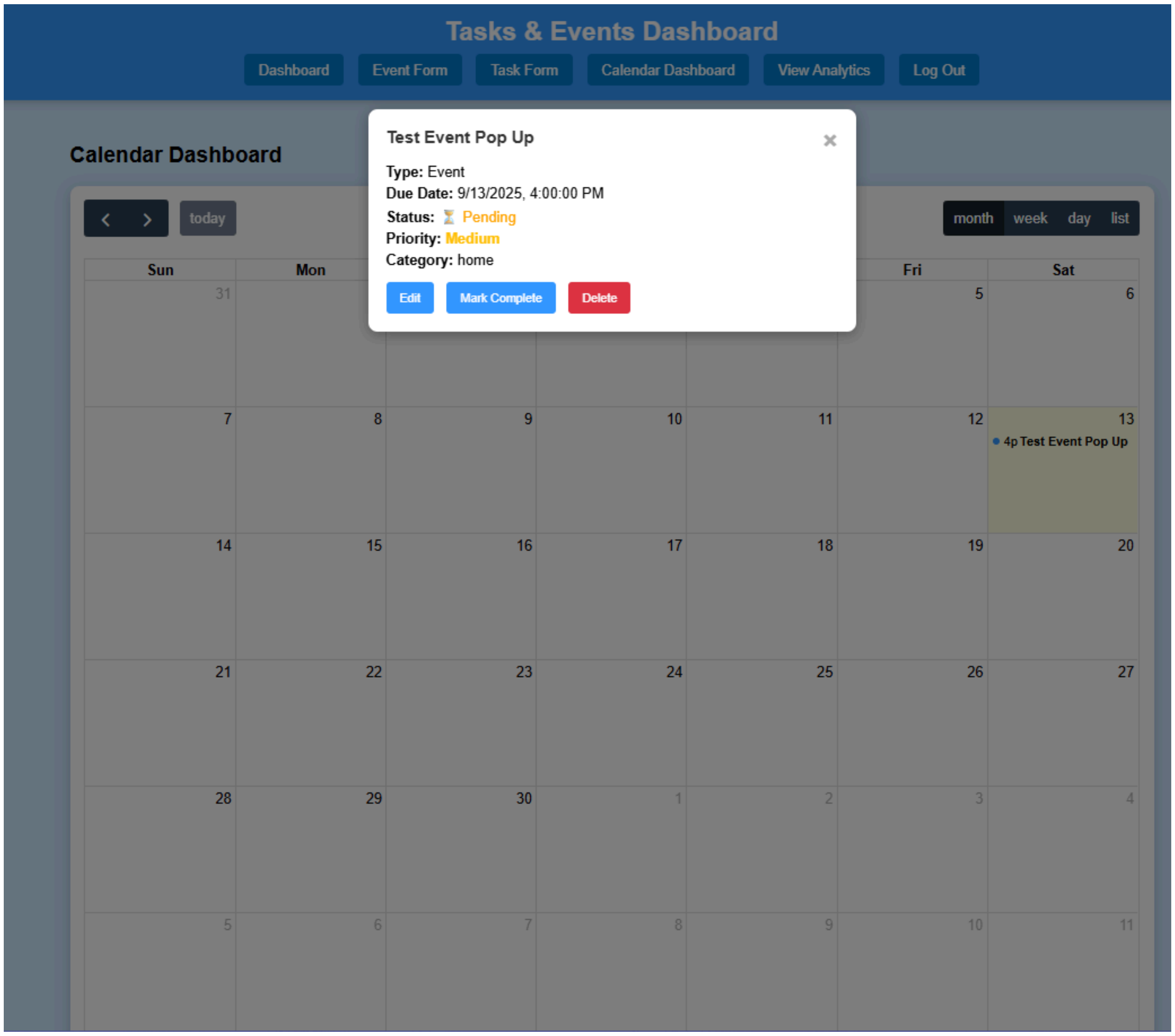
Event Color

Choose a color for this event in the calendar view.

Add Event

Calendar Dashboard

- **Full Calendar View**: Monthly, weekly, daily, and list views
- **Interactive Events**: Click to view/edit details
- **Drag & Drop**: Move events by dragging
- **Quick Create**: Click empty date to create event
- **Event Details**: View complete information in popup

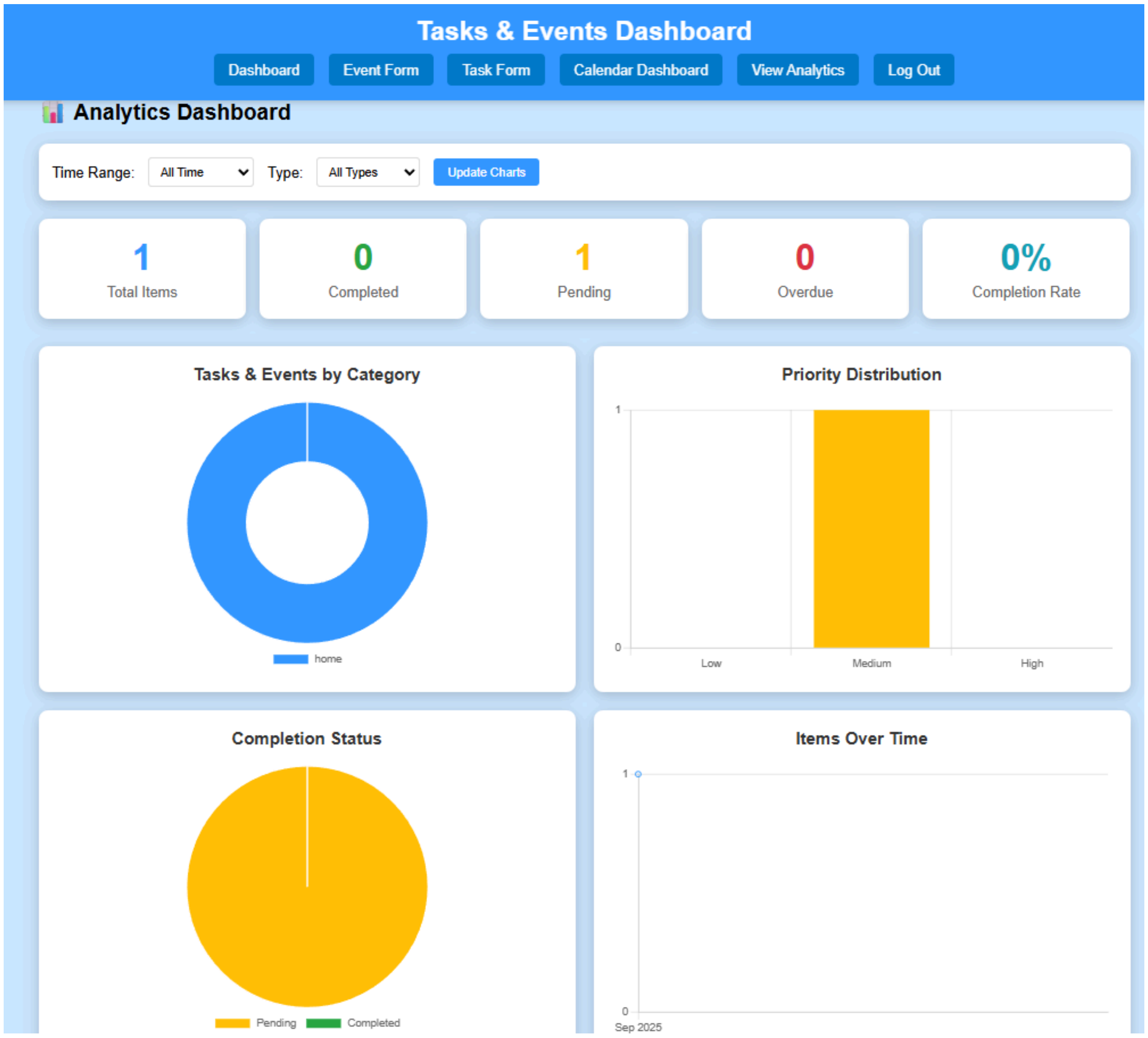


Test Event Pop Up

Type: Event
Due Date: 9/13/2025, 4:00:00 PM
Status: ⌛ Pending
Priority: Medium
Category: home

Analytics Dashboard

- **Statistics**: Total, completed, pending, overdue counts
- **Charts**: Category distribution, priority breakdown, completion trends
- **Filters**: Time range and type filtering
- **Auto-refresh**: Updates every 30 seconds



Analytics Dashboard

Time Range: All Time Type: All Types Update Charts

1	0	1	0	0%
Total Items	Completed	Pending	Overdue	Completion Rate

Troubleshooting

Forgotten Username/Password

SQL Database Query

In the terminal, use the following queries:

To see all usernames:
```
sqlite3 tasks.db "SELECT username FROM users;"
```

To see usernames with creation dates:
```
sqlite3 tasks.db "SELECT username, created_at FROM users;"
```

To see everything including password hashes:
```
sqlite3 tasks.db "SELECT * FROM users;"
```

To find a specific user:
```
sqlite3 tasks.db "SELECT username, password_hash FROM users WHERE
username = 'your_username';"
```

Reset Password

Update the hash directly using the following command:
```
python -c "from werkzeug.security import generate_password_hash;
print(generate_password_hash('newpassword'))"
```

Then, update the database:
```
sqlite3 tasks.db "UPDATE users SET password_hash = 'new_hash_here'
WHERE username = 'your_username';"
```

Common Issues

Bot not responding to commands:

- Check if bot is in correct channel
- Verify DISCORD_COMMANDS_CHANNEL_ID is correct
- Ensure bot has proper permissions

Web app not accessible:

- Check if port 5000 is available
- Try http://localhost:5000 instead
- Verify no firewall blocking

Database errors:

- Run `python setup.py` to reinitialize
- Check file permissions on tasks.db
- Ensure Python has write access to directory

Discord notifications not working:

- Verify DISCORD_NOTIFICATIONS_CHANNEL_ID is correct
- Check bot permissions in notification channel
- Confirm bot is connected (check console output)

Manual Testing

- Visit `/status` endpoint to check system health
- Use `/trigger_daily_update` to test notification system
- Check console output for detailed error messages

Support

- All errors are logged to console with timestamps
- Database includes migration system for updates
- Configuration validation on startup prevents common issues