**Panchagam Telegram Bot: Complete Documentation**

**Table of Contents**

1. [Architecture Overview](#architecture-overview)
2. [Technical Design](#technical-design)
3. [Setup and Configuration](#setup-and-configuration)
4. [Database Structure](#database-structure)
5. [Code Organization](#code-organization)
6. [Key Features](#key-features)
7. [Notification System](#notification-system)
8. [Anti-Sleep Mechanism](#anti-sleep-mechanism)
9. [Webhook Implementation](#webhook-implementation)
10. [User Preference Management](#user-preference-management)
11. [Deployment Guide](#deployment-guide)
12. [Frequent Issues and Solutions](#frequent-issues-and-solutions)
13. [Supabase Integration](#supabase-integration)
14. [Maintenance Guide](#maintenance-guide)
15. [Future Enhancements](#future-enhancements)

**Architecture Overview**

The Panchagam Telegram Bot is a Node.js application that provides daily Hindu astrological calendar information and timely notifications. The architecture follows a webhook-based model deployed on Render's cloud platform.

**Components:**

* **Telegram Bot**: Handles user interactions and sends notifications
* **Express Server**: Hosts the webhook endpoint and serves a status page
* **Supabase Database**: Stores the Panchagam data
* **File System Storage**: Manages user preferences with persistence
* **Cron Jobs**: Schedules notifications and maintenance tasks
* **Self-Polling System**: Prevents the app from sleeping on Render's free tier

**Architecture Diagram:**

┌────────────────┐ ┌─────────────────┐

│ Telegram │◄───────►│ Express Server │

│ │ API │ (Webhook Mode) │

└────────────────┘ └────────┬─────────┘

│

│

┌────────────────┐ │ ┌─────────────────┐

│ Cron Jobs & │◄───────┼───────►│ Supabase DB │

│ Notifications │ │ │ │

└────────────────┘ │ └─────────────────┘

│

│

┌────────────────┐ │ ┌─────────────────┐

│ Self-Polling │◄───────┼───────►│ User Preferences│

│ (Anti-Sleep) │ │ │ (File Storage) │

└────────────────┘ │ └─────────────────┘

▼

┌─────────────────┐

│ Status Page & │

│ Health Endpoints│

└─────────────────┘

**Technical Design**

**1. Framework Selection**

* **Telegraf.js**: Chosen for its modern API and webhook support
* **Express.js**: Lightweight server for handling webhook requests
* **Node-cron**: For scheduling notifications
* **Moment-timezone**: For handling timezone conversions (Asia/Kolkata)
* **Supabase-js**: SDK for connecting to Supabase database
* **Axios**: For making HTTP requests in the self-polling mechanism

**2. Deployment Platform**

* **Render**: Cloud platform with free tier capabilities
* **Webhook Mode**: More efficient than polling for cloud deployment
* **Persistent Disk**: For storing user preferences between restarts

**3. Design Patterns**

* **Event-based Architecture**: Bot responds to Telegram events via webhook
* **Observer Pattern**: Notification system watches for upcoming events
* **Repository Pattern**: Database access is abstracted through helper functions
* **Singleton Pattern**: Single bot instance and database connection

**Setup and Configuration**

**Environment Variables**

BOT\_TOKEN=your\_telegram\_bot\_token

SUPABASE\_URL=your\_supabase\_project\_url

SUPABASE\_KEY=your\_supabase\_api\_key

APP\_URL=your\_render\_app\_url

NODE\_ENV=production

**Dependencies**

json

{

"dependencies": {

"express": "^4.18.2",

"node-cron": "^3.0.2",

"telegraf": "^4.12.2",

"@supabase/supabase-js": "^2.38.4",

"moment-timezone": "^0.5.43",

"axios": "^1.6.2"

}

}

**Database Structure**

**Table: daily\_panchangam**

The Supabase database contains a daily\_panchangam table with the following structure:

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID | Primary key |
| date | DATE | Date for the panchagam data |
| vaara | TEXT | Day of the week in Tamil |
| sunrise | TIMESTAMPTZ | Sunrise time |
| sunset | TIMESTAMPTZ | Sunset time |
| moonrise | TIMESTAMPTZ | Moonrise time |
| moonset | TIMESTAMPTZ | Moonset time |
| nakshatra | JSON | Array of nakshatra data |
| tithi | JSON | Array of tithi data |
| karana | JSON | Array of karana data |
| yoga | JSON | Array of yoga data |
| main\_nakshatra | TEXT | Primary nakshatra |
| is\_amavasai | BOOLEAN | New moon indicator |
| is\_pournami | BOOLEAN | Full moon indicator |
| is\_valar\_pirai | BOOLEAN | Waxing moon indicator |
| is\_thei\_pirai | BOOLEAN | Waning moon indicator |
| chandrashtama\_for | JSON | Array of affected nakshatras |
| cosmic\_score | NUMERIC | Auspiciousness score |
| rahu\_kalam | TEXT | Rahu Kalam time range (format: "HH:MM AM - HH:MM PM") |
| yamagandam | TEXT | Yamagandam time range |
| kuligai | TEXT | Kuligai time range |
| abhijit\_muhurta | TEXT | Abhijit Muhurta time range |
| updated\_at | TIMESTAMPTZ | Last update timestamp |

**JSON Column Structures**

**nakshatra format:**

json

[

{

"id": 13,

"name": "சித்திரை",

"lord": {

"id": 4,

"name": "செவ்வாய்",

"vedic\_name": "செவ்வாய்"

},

"start": "2025-04-12T18:07:53+05:30",

"end": "2025-04-13T21:10:52+05:30"

}

]

**tithi format:**

json

[

{

"id": 32,

"index": 0,

"name": "பிரதமை",

"paksha": "கிருஷ்ண பக்ஷ",

"start": "2025-04-13T05:52:16+05:30",

"end": "2025-04-14T08:25:31+05:30"

}

]

**chandrashtama\_for format:**

json

["Purva Bhadrapada", "Shatabhisha"]

**Code Organization**

The code is organized into the following major sections:

1. **Initialization and Configuration**: Setting up Express, Telegraf, Supabase
2. **User Preference Management**: Loading/saving user preferences
3. **Helper Functions**: Time formatting, JSON parsing, data retrieval
4. **Command Handlers**: Bot command implementations
5. **Notification System**: Scheduled notifications for various events
6. **Self-Polling Mechanism**: Keeping the app awake
7. **Web Server**: Status page and health checks
8. **Webhook Setup**: Telegram webhook configuration

**Key Features**

**1. Daily Panchagam Information**

* Today's and tomorrow's astrological information
* Clean, well-formatted message display
* Support for Tamil astrological terms

**2. Notification System**

* Daily morning summary at 6:00 AM
* Alerts 15 minutes before Rahu Kalam
* Alerts 15 minutes before Yamagandam
* Alerts 15 minutes before Kuligai
* Alerts 15 minutes before Abhijit Muhurta

**3. User Preferences**

* Toggle notifications for specific events
* Persistent storage across app restarts
* Settings management through Telegram interface

**4. Admin Functions**

* /stats command shows bot usage statistics
* /test command tests database connection
* Status page shows uptime and user count

**5. Anti-Sleep Mechanism**

* Self-polling to prevent Render free tier from sleeping
* 40-second ping interval (below Render's 50-second timeout)
* Health check endpoints for external monitoring

**Notification System**

The notification system uses node-cron to schedule various types of notifications:

**Daily Morning Summary (6:00 AM)**

javascript

cron.schedule('0 6 \* \* \*', async () => {

*// Sends today's panchagam to all users with daily notifications enabled*

}, { timezone: TIMEZONE });

**Period Notifications (Every 5 Minutes)**

javascript

cron.schedule('\*/5 \* \* \* \*', async () => {

*// Checks if any periods (Rahu Kalam, etc.) start in 15 minutes*

*// Sends notifications to users who have enabled them*

}, { timezone: TIMEZONE });

**Self-Check (Every 10 Minutes)**

javascript

cron.schedule('\*/10 \* \* \* \*', () => {

*// Logs stats for monitoring*

}, { timezone: TIMEZONE });

**Notification Format**

All notifications use Markdown formatting with emojis and follow this structure:

🌞 \*Good Morning! Here's your daily Panchagam update:\*

📅 \*DAILY PANCHAGAM - {date} ({day})\*

...

**Anti-Sleep Mechanism**

Render's free tier puts applications to sleep after 50 seconds of inactivity. The bot implements a self-polling mechanism to prevent this:

javascript

async function setupSelfPolling() {

*// Pings itself every 40 seconds*

const pingInterval = 40 \* 1000;

const pingServer = async () => {

*// Make HTTP request to own /ping endpoint*

*// Schedule next ping*

setTimeout(pingServer, pingInterval);

};

*// Start the ping cycle*

setTimeout(pingServer, pingInterval);

}

This ensures the app stays active 24/7, even on Render's free tier.

**Webhook Implementation**

The bot uses Telegram's webhook mechanism instead of polling for better performance on cloud platforms:

javascript

*// Set up webhook mode*

const secretPath = `/telegraf/${bot.secretPathComponent()}`;

app.use(bot.webhookCallback(secretPath));

*// Configure webhook URL*

const webhookUrl = `${appUrl}${secretPath}`;

await bot.telegram.setWebhook(webhookUrl);

This approach:

1. Creates a secure, random endpoint
2. Sets it up with Telegram's API
3. Avoids the polling conflicts that can occur with multiple instances

**User Preference Management**

User preferences are stored in a JSON file (data/preferences.json) and managed with these functions:

**Loading Preferences**

javascript

let userPreferences = {};

try {

if (fs.existsSync(PREFS\_FILE)) {

const data = fs.readFileSync(PREFS\_FILE, 'utf8');

userPreferences = JSON.parse(data);

}

} catch (error) {

console.error('Error loading preferences:', error);

}

**Saving Preferences**

javascript

function savePreferences() {

try {

fs.writeFileSync(PREFS\_FILE, JSON.stringify(userPreferences, null, 2), 'utf8');

} catch (error) {

console.error('Error saving preferences:', error);

}

}

**Preference Structure**

json

{

"user\_id": {

"notifyRahuKalam": true,

"notifyYamagandam": true,

"notifyChandrashtama": true,

"notifyDaily": true

}

}

**Deployment Guide**

**Initial Setup on Render**

1. Create a new Web Service in Render
2. Connect to your GitHub repository
3. Configure service:
   * **Name**: panchagam-telegram-bot
   * **Environment**: Node.js
   * **Build Command**: npm install
   * **Start Command**: node index.js
   * **Environment Variables**: Set BOT\_TOKEN, SUPABASE\_URL, SUPABASE\_KEY, APP\_URL, NODE\_ENV
   * **Create Disk**: Mount path: /data, Size: 1GB

**Monitoring and Maintenance**

1. Set up Uptime Robot to monitor /ping endpoint
2. Check Render logs for any errors
3. Use the built-in status page to monitor uptime

**Frequent Issues and Solutions**

**1. Bot Not Responding**

**Symptoms:**

* Bot doesn't respond to messages
* No errors in logs

**Solutions:**

* Check if webhook is set correctly: https://api.telegram.org/bot<BOT\_TOKEN>/getWebhookInfo
* Verify the BOT\_TOKEN environment variable
* Restart the service on Render

**2. Database Connection Issues**

**Symptoms:**

* "Database connection failed" errors in logs
* Bot responds but can't retrieve Panchagam data

**Solutions:**

* Check SUPABASE\_URL and SUPABASE\_KEY environment variables
* Verify the table name (should be daily\_panchangam)
* Check if the table structure matches expected schema
* Test with /test command

**3. Missing Notifications**

**Symptoms:**

* Bot works but notifications aren't sent
* No errors in logs related to notifications

**Solutions:**

* Check user preferences with /myprefs command
* Verify that cron jobs are running (check logs for "Running period notification check")
* Ensure the app isn't sleeping (check logs for "Self-ping successful")
* Test with /testdaily and /testrahu commands

**4. Sleep Issues with Render**

**Symptoms:**

* Bot goes offline periodically
* Long gaps in logs

**Solutions:**

* Verify APP\_URL environment variable is set correctly
* Check logs for "Pinging self" messages
* Set up external monitoring with Uptime Robot
* Upgrade to paid tier if necessary

**Supabase Integration**

**Connection Setup**

javascript

const supabase = createClient(

process.env.SUPABASE\_URL,

process.env.SUPABASE\_KEY

);

**Data Retrieval**

javascript

const getPanchagamForDate = async (date) => {

const formattedDate = moment(date).format('YYYY-MM-DD');

try {

const { data, error } = await supabase

.from(TABLE\_NAME)

.select('\*')

.eq('date', formattedDate);

if (error) {

console.error('Database query error:', error);

return null;

}

if (!data || data.length === 0) {

console.log(`No data found for date: ${formattedDate}`);

return null;

}

return data[0];

} catch (e) {

console.error('Exception in getPanchagamForDate:', e);

return null;

}

};

**Common Supabase Issues**

1. **Permission Issues**:
   * Ensure the API key has the necessary permissions
   * For read-only operations, the anon key is sufficient
   * For writing data, you may need a service role key
2. **Rate Limiting**:
   * Free tier has limits on database operations
   * Implement caching if necessary for high-traffic periods
3. **JSON Parsing**:
   * Supabase may return JSON fields as strings or objects
   * Use the safelyParseJSON helper function to handle both cases
4. **Connection Pooling**:
   * Supabase may have connection limits on free tier
   * The bot reuses a single connection for all operations

**Maintenance Guide**

**Regular Maintenance Tasks**

1. **Database Backup**:
   * Regularly export Panchagam data from Supabase
   * Back up user preferences from the data volume
2. **Log Review**:
   * Check Render logs for errors or warnings
   * Monitor notification success/failure rates
3. **Bot Updates**:
   * Keep dependencies updated
   * Test on a staging environment before deploying

**Adding New Panchagam Data**

1. Insert new records into the daily\_panchangam table with the correct format
2. Ensure all required fields are populated
3. JSON fields (nakshatra, tithi, etc.) should follow the documented structure

**Scaling Considerations**

1. **User Growth**:
   * Monitor user count with /stats command
   * Consider upgrading to paid tier if approaching free tier limits
2. **Data Volume**:
   * Panchagam data should be added in advance
   * Consider implementing pagination if data volume grows significantly

**Future Enhancements**

1. **Multi-language Support**:
   * Add option to switch between Tamil and English
2. **Advanced Notifications**:
   * Allow users to customize notification timing
   * Add weekly summary option
3. **Calendar Integration**:
   * Export auspicious times to Google Calendar
   * Add iCal format support
4. **Extended Information**:
   * Add detailed explanations of astrological elements
   * Include prayer recommendations
5. **User Analytics**:
   * Track feature usage
   * Gather feedback through bot interface

This documentation provides a comprehensive overview of the Panchagam Telegram Bot's architecture, implementation, and maintenance procedures. It should serve as a complete reference for anyone taking over development or management of the bot.