Sports Bar TV Controller - System Documentation

Version: 2.3

Last Updated: October 17, 2025

Status: Production Ready

MUST READ DOCUMENTATION - ESSENTIAL REFERENCE MATERIALS

CRITICAL: Before making ANY changes to the system, especially UI/styling changes, you MUST review these essential documentation files located in the docs/ directory:

Color Scheme & Styling Standards

- COLOR_SCHEME_STANDARD.md Official color palette and usage guidelines
- COLOR_SCHEME_STANDARD.pdf PDF version of color standards
- COLOR_STANDARDIZATION_SUMMARY.md Summary of color standardization implementation
- COLOR_STANDARDIZATION_SUMMARY.pdf PDF version of standardization summary

System Architecture & Performance

- SUBSCRIPTION_POLLING_IMPLEMENTATION.md Subscription polling system details
- SUBSCRIPTION_POLLING_IMPLEMENTATION.pdf PDF version of polling implementation
- SYSTEM OPTIMIZATION SUMMARY.md System-wide optimization documentation

SSH Connection & Remote Access

- **SSH_OPTIMIZATION_GUIDE.md** Comprehensive guide for efficient SSH connections to remote server
- Solves SSH connection hanging issues
- Provides optimized connection methods (heredoc, SSH config, control sockets)
- Essential for automation and reliable remote server operations
- Includes connection multiplexing and performance optimization techniques

Why These Documents Matter

These documents contain:

- Approved color palettes that ensure readability and consistency
- Component styling standards that prevent UI regressions
- Performance optimization guidelines that maintain system efficiency
- Implementation patterns that have been tested and validated

MARNING: Ignoring these standards can result in:

- Unreadable text and poor contrast (Q&A page issues)
- Broken UI components (Matrix page display issues)
- Reverted changes (Global Cache device page problems)
- Inconsistent user experience across the application

Documentation Location: /home/ubuntu/Sports-Bar-TV-Controller-local/docs/ (loc-

al) or /root/sports-bar-tv-controller/docs/ (production server)

Quick Access Information

Server Access

Host: 24.123.87.42Port: 224 (SSH)

• Application Port: 3000 (HTTP)

• Username: ubuntu

Password: 6809233DjD\$\$\$ (THREE dollar signs)Application URL: http://24.123.87.42:3000

SSH Connection:

ssh -p 224 ubuntu@24.123.87.42

GitHub Repository

- Repository: https://github.com/dfultonthebar/Sports-Bar-TV-Controller
- **Development Project Path:** /home/ubuntu/github_repos/Sports-Bar-TV-Controller
- **Production Server Path:** /home/ubuntu/Sports-Bar-TV-Controller
- PM2 Process Name: sports-bar-tv
- GitHub Token: Stored securely in server environment (not documented for security)

Quick Deployment to Production

```
# SSH into production server
ssh -p 224 ubuntu@24.123.87.42
# Navigate to project (PRODUCTION PATH)
cd /home/ubuntu/Sports-Bar-TV-Controller
# Pull latest changes
git pull origin main
# Install dependencies
npm install
# Generate Prisma Client
npx prisma generate
# Build application
npm run build
# Restart application (PM2 process name: sports-bar-tv)
pm2 restart sports-bar-tv
# Check logs
pm2 logs sports-bar-tv
```

AI System Access

IMPORTANT: This section contains credentials for AI assistants to access and manage the system autonomously. These credentials are intentionally documented here for future AI access.

System Access Credentials

• Web Interface: http://24.123.87.42:3000/

• **SSH Host:** 24.123.87.42

• SSH Port: 224

• SSH Username: ubuntu

• **SSH Password:** 6809233DjD\$\$\$ (THREE dollar signs)

• Project Directory: /home/ubuntu/Sports-Bar-TV-Controller

• GitHub Repository: https://github.com/dfultonthebar/Sports-Bar-TV-Controller

Device Information

• Fire TV Cube IP Address: 192.168.5.131

- Used for ADS bridge configuration
- Connected to network for remote control
- Integration with matrix switcher system

SSH Connection Command

```
ssh -p 224 ubuntu@24.123.87.42
```

Quick Access for AI Assistants

1. Access System:

bash

ssh -p 224 ubuntu@24.123.87.42 # Password: 6809233DjD\$\$\$

2. Navigate to Project:

bash

cd /home/ubuntu/Sports-Bar-TV-Controller

3. View Application:

- Open browser to: http://24.123.87.42:3000/

4. Manage Services:

```bash

# View logs

pm2 logs sports-bar-tv

# Restart application

pm2 restart sports-bar-tv

# Check status

pm2 status

...

# **Notes for AI Assistants**

- Password has THREE dollar signs at the end: 6809233DjD\$\$\$
- System runs on Intel NUC13ANHi5 hardware
- Application managed via PM2 process manager
- Database: PostgreSQL with Prisma ORM
- Web framework: Next.js 14 with TypeScript
- · Always pull latest changes before making modifications
- Use PM2 to restart after code updates

# **Database & Prisma Setup**

# **Database Configuration**

• Type: PostgreSQL

• Connection: Configured in . env file

• ORM: Prisma

# **Prisma Commands**

```
Generate Prisma Client
npx prisma generate

Run migrations
npx prisma migrate dev

Deploy migrations (production)
npx prisma migrate deploy

Open Prisma Studio (database browser)
npx prisma studio

Check migration status
npx prisma migrate status
```

# **Database Schema Location**

Schema File: prisma/schema.prismaMigrations: prisma/migrations/

# **Key Database Models**

- MatrixOutput TV display outputs
- MatrixInput Video sources
- WolfpackConfig Matrix switcher configuration
- AudioProcessor Atlas audio configuration
- IndexedFile Al Hub codebase files
- QAPair Al Hub Q&A training data
- TrainingDocument Al Hub training documents
- ApiKey Al provider API keys
- T0D0 Task management

# **System Overview**

The Sports Bar TV Controller is a comprehensive web application designed to manage TV displays, matrix video routing, and sports content scheduling for sports bar environments.

# **Technology Stack**

• Frontend: Next.js 14, React, TypeScript, Tailwind CSS

• Backend: Next.js API Routes, Prisma ORM

Database: PostgreSQLHardware Integration:

• Wolfpack HDMI Matrix Switchers (via HTTP API)

• Atlas AZMP8 Audio Processor (via HTTP API)

• Process Management: PM2

• Al Integration: Multiple Al providers (Ollama, Abacus Al, OpenAl, Anthropic, X.Al)

# **Application Features by Tab**

# 1. Dashboard (Home)

### **Overview**

Main landing page providing quick access to all system features and current system status.

### **Features**

- System Status "Server Online" indicator with operational status
- Quick Access Cards:
- Al Hub Unified Al management & assistance
- Sports Guide Find where to watch sports
- Remote Control Control TVs and audio systems
- System Admin Logs, backups, sync & tests

# **Navigation**

- Direct access to all major subsystems
- System health indicators
- Recent activity display

# **API Endpoints**

• N/A (frontend only)

# 2. Video Matrix / Matrix Control

# **Overview**

Comprehensive video routing system for managing HDMI matrix switchers and TV displays.

### **Features**

# **Output Configuration**

- Outputs 1-4 (TV 01-04): Full matrix outputs with complete controls
- Power on/off toggle button (green when on, gray when off)
- Active/inactive checkbox
- Label field (TV 01, TV 02, TV 03, TV 04)
- Resolution dropdown (1080p, 4K, 720p)
- · Audio output field
- Full Wolfpack integration
- Outputs 5-32: Regular matrix outputs with full controls
- Outputs 33-36 (Matrix 1-4): Audio routing outputs with special controls
- Used for Atlas audio processor integration
- · Video input selection affects audio routing

# **Input Configuration**

- Configure 32 video sources
- Custom labeling (e.g., "Cable Box 1", "Apple TV")
- Enable/disable individual inputs

# **TV Selection System**

Granular control over which TVs participate in automated schedules:

- dailyTurnOn Boolean flag for morning schedule participation
- dailyTurnOff Boolean flag for "all off" command participation
- Configured per output in the database

# **API Endpoints**

# GET/POST /api/matrix/outputs

- GET: Retrieve all matrix outputs
- POST: Update output configuration
- **Body**: { outputNumber, label, enabled, dailyTurnOn, dailyTurnOff }

# GET /api/matrix/outputs-schedule

Retrieve outputs with schedule participation flags

# POST /api/matrix/route

Route a source to an output:

```
{
 "input": 5,
 "output": 33
}
```

# POST /api/matrix/power

Control output power:

```
{
 "output": 33,
 "state": "on" // or "off"
}
```

### POST /api/matrix/video-input-selection

Route video input to Matrix 1-4 audio outputs (33-36)

# **Database Schema**

# **MatrixOutput**

# MatrixInput

# **Troubleshooting**

### **Matrix Switching Not Working:**

- 1. Test connection in System Admin
- 2. Verify output/input configuration
- 3. Check Wolfpack matrix is powered on
- 4. Verify network connectivity
- 5. Test individual commands

# **TV Selection Not Working:**

- 1. Verify database migration status
- 2. Check output configuration flags
- 3. Restart application

# 3. Atlas / Audio Control

### Overview

Multi-zone audio control system with Atlas AZMP8 processor integration.

#### **Features**

# **Atlas AZMP8 Configuration**

• IP Address: 192.168.5.101:80

- Model: AZMP8 (8 inputs, 8 outputs, 8 zones)
- Status: Online and authenticated

# **Configured Audio System**

# 7 Inputs:

- Matrix 1-4 (video input audio)
- Mic 1-2
- Spotify

### 7 Outputs/Zones:

- Bar
- Bar Sub
- Dining Room
- Party Room West
- Party Room East
- Patio
- Bathroom

**3 Scenes:** Preset configurations for different scenarios

# **Dynamic Zone Labels**

- Zone labels update automatically based on selected video input
- When video input is selected for Matrix 1-4, zone labels reflect the input name
- Example: Selecting "Cable Box 1" updates zone label from "Matrix 1" to "Cable Box 1"
- Falls back to "Matrix 1-4" when no video input selected

#### **Features**

- Real-time zone control
- Volume adjustment per zone
- Input selection per zone
- Scene management
- Configuration upload/download
- Automatic timestamped backups

# **API Endpoints**

# **GET** /api/audio-processor

Get all configured audio processors

# POST /api/atlas/upload-config

Upload configuration to Atlas processor

# **GET** /api/atlas/download-config

Download current configuration from Atlas processor

### **POST** /api/atlas/route-matrix-to-zone

Route audio from matrix output to zone

# **GET** /api/atlas/ai-analysis

Get Al-powered analysis of audio system performance

# **Configuration Management**

# **Configuration File Location:**

- Primary: /home/ubuntu/github\_repos/Sports-Bar-TV-Controller/data/atlas-configs/cmgjx-a5ai000260a7xuiepjl.json
- Backups: /home/ubuntu/github\_repos/Sports-Bar-TV-Controller/data/atlas-configs/cmgjx-a5ai000260a7xuiepjl backup \*.json

### **Backup Strategy:**

- Automatic backup created on every upload
- Timestamped filename format
- Manual restore by copying backup to primary config file

# **Database Schema**

# **Troubleshooting**

#### **Atlas Shows Offline:**

- 1. Check network connectivity: ping 192.168.5.101
- 2. Verify configuration file exists
- 3. Check processor is powered on
- 4. Restore from backup if needed

### **Configuration Not Loading:**

- 1. Validate JSON configuration file
- 2. Check file permissions
- 3. Restore from most recent backup

# 4. Al Hub

# **Overview**

Unified AI management system providing intelligent assistance, codebase analysis, device insights, and AI configuration.

### **Current Status**

Testing Date: October 15, 2025

Overall Status: A PARTIALLY FUNCTIONAL

**Critical Issues:** 2 **Features Tested:** 7

Working Features: 5
Broken Features: 2

# Features & Status

# Al Assistant Tab (Partially Working)

Status: Chat interface works, Codebase sync fails

#### **Chat Interface:**

- V Status: WORKING

- **Performance Issue:** Response time is slow (15+ seconds)

- Functionality: Successfully answers questions about the codebase

- Features:

- Natural language queries

- Codebase context awareness

- Troubleshooting assistance
- Code explanations

#### **Sync Codebase:**

- X Status: FAILING

- Error: GET http://24.123.87.42:3000/api/ai-assistant/index-codebase 404 (Internal Server

Error)

- Impact: Cannot index codebase for AI analysis

- Priority: CRITICAL - Fix immediately

# ▼ Teach AI Tab (UI Works, Backend Fails)

### **Upload Documents:**

- **W** UI Status: WORKING

- **Supported Formats:** PDF, Markdown (.md), Text (.txt)

- Features:

- Drag and drop file upload

- Multiple file support

- File type validation

- Note: Upload errors observed, needs further testing

# **Q&A Training:**

- X Status: FAILING

- Error: Database error: Failed to create Q&A entry

- Console Error: 500 (Internal Server Error) for api/qa-entries.ts

- Impact: Users cannot add Q&A training pairs

- Priority: CRITICAL - Fix immediately

- Features (Non-functional):

- Category selection (General, Technical, Troubleshooting, etc.)
- Question/Answer input fields
- Entry management
- Generate from Repository
- Generate from Docs
- Upload Q&A File

#### Test AI:

- **W UI Status:** WORKING

- Features:

- Test question input
- AI response testing
- Testing tips and guidance
- Note: Cannot fully test without training data

### **Statistics Display:**

- Documents: 0 - O&A Pairs: 0

- Total Content: 0 Bytes

- Last Updated: 10/15/2025, 1:00:06 AM

# Enhanced Devices Tab (Working)

Status: V FULLY FUNCTIONAL

#### **Features:**

- Device AI Assistant for intelligent insights
- Filter options:
- All Devices dropdown
- Time range filter (Last 24 Hours)
- Refresh button
- Tabs:
- Smart Insights
- Performance
- Recommendations
- Predictions
- Current State: "No Al insights available for the selected criteria"

# Configuration Tab (Working)

Status: V FULLY FUNCTIONAL

### **Provider Statistics:**

- 1 Active Local Service
- 3 Cloud APIs Ready
- 5 Inactive Local Services

#### **Local AI Services:**

- V Ollama (http://localhost:11434/api/tags, Model: phi3:mini) Active (4ms)
- X Custom Local AI (http://localhost:8000/v1/models) Error
- X LocalAI (http://localhost:8080/v1/models) Error
- X LM Studio (http://localhost:1234/v1/models) Error
- X Text Generation WebUI (http://localhost:5000/v1/models) Error
- X Tabby (http://localhost:8080/v1/models) Error

### **Cloud AI Services:**

- **OpenAI** Ready (API key configured)
- **Anthropic Claude** Ready (API key configured)
- X.AI Grok Ready (API key configured)
- Abacus AI Not Configured (No API key)

# Features:

- Al System Diagnostics (expandable)
- Provider status monitoring

- Refresh status button
- Local AI setup guide

# API Keys Tab (Working)

Status: V FULLY FUNCTIONAL

#### **Features:**

- API key management interface
- Configured API Keys display (currently 0)
- Add API Key button
- Provider documentation links:
- Ollama (Local) RECOMMENDED
- Abacus Al
- OpenAl
- LocalAI
- Custom Local AI
- Local AI Services Status:
- Port 8000: Active (Custom service detected)
- Port 11434: Check if Ollama is running
- Port 8080: Check if LocalAI is running

### **AI Assistant Features Listed:**

- Equipment Troubleshooting
- System Analysis
- Configuration Assistance
- Sports Guide Intelligence
- Operational Insights
- Proactive Monitoring

# **Database Schema**

```
model IndexedFile {
 id
 @id @default(cuid())
 String
 filePath
 String
 @unique
 fileName
 String
 fileType
 String
 content
 String
 @db.Text
 fileSize
 Int
 lastModified DateTime
 lastIndexed DateTime @default(now())
 hash String
isActive Boolean @default(true)
createdAt DateTime @default(now())
updatedAt DateTime @updatedAt
}
model QAPair {
 id String @id @defa
question String @db.Text
answer String @db.Text
context String? @db.Text
source String?
category String?
 @id @default(cuid())
 isActive Boolean @default(true)
 createdAt DateTime @default(now())
 updatedAt DateTime @updatedAt
}
model TrainingDocument {
 @id @default(cuid())
 id
 String
 title
 String
 content
 String
 @db.Text
 fileType String
 fileSize
 Int
 isActive
 String?
 Boolean @default(true)
 createdAt
updatedAt
DateTime @default(now())
DateTime @updatedAt
model ApiKey {
 @id @default(cuid())
 id
 String
 provider
 String
 keyName
 String
 apiKey
 String
 isActive
 Boolean @default(true)
 createdAt DateTime @default(now())
 @@unique([provider, keyName])
}
```

# **API Endpoints**

POST /api/ai-assistant/index-codebase

X Status: BROKEN (404 error)
Index codebase files for Al analysis

# POST /api/ai-assistant/chat

✓ **Status:** WORKING (slow) Chat with Al about codebase

# **POST** /api/ai/qa-generate

Generate Q&A pairs from repository

# POST /api/ai/qa-entries

X Status: BROKEN (500 error)
Create Q&A training entries

# **GET/POST** /api/api-keys

▼ Status: WORKING

Manage AI provider API keys

# **POST** /api/devices/ai-analysis

Get AI insights for devices

### Critical Issues & Fix Plan

# CRITICAL #1: Q&A Training Database Error

**API:** POST /api/ai/qa-entries returns 500 error **Impact:** Users cannot add Q&A training pairs

#### **Fix Steps:**

- 1. Check database schema for QAPair table
- 2. Verify Prisma migrations are up to date
- 3. Review API route handler ( src/app/api/ai/qa-entries/route.ts )
- 4. Check database connection and write permissions
- 5. Add proper error logging
- 6. Test with various Q&A entry formats

Priority: Fix immediately before production use

# CRITICAL #2: Codebase Indexing 404 Error

**Error:** GET http://24.123.87.42:3000/api/ai-assistant/index-codebase 404

Impact: Cannot index codebase for AI assistance

#### Fix Steps:

- 1. Verify API route exists in correct location
- 2. Check route file naming (should be route.ts in app router)
- 3. Ensure proper HTTP method handling (GET/POST)
- 4. Implement codebase indexing logic if missing
- 5. Test with actual project directory
- 6. Add proper error responses

Priority: Fix immediately for full AI Hub functionality

# HIGH PRIORITY: Chat Performance

**Issue:** 15+ second response time **Impact:** Poor user experience

#### **Optimization Steps:**

- 1. Profile AI model response time
- 2. Implement streaming responses
- 3. Add response caching for common questions
- 4. Consider faster AI model for simple queries
- 5. Optimize context window size
- 6. Add better loading indicators



### MEDIUM PRIORITY: Local AI Services

**Issue:** 5 local Al services showing error status

#### Services to Fix:

- Custom Local AI (port 8000)
- LocalAI (port 8080)
- LM Studio (port 1234)
- Text Generation WebUI (port 5000)
- Tabby (port 8080 port conflict?)

#### Fix Steps:

- 1. Verify each service is installed
- 2. Check if services are running
- 3. Update service URLs in configuration
- 4. Add health check with retry logic
- 5. Document installation instructions
- 6. Consider making local services optional

# Recommendations

# **Immediate Actions:**

- 1. Fix Q&A Training database error (CRITICAL)
- 2. Fix Codebase Indexing 404 error (CRITICAL)
- 3. Test document upload feature thoroughly
- 4. Add proper error messages and user feedback

#### **Short-term Improvements:**

- 1. Optimize chat response performance
- 2. Implement streaming responses
- 3. Add progress indicators
- 4. Configure local AI services

# **Long-term Enhancements:**

- 1. Add training data export/import
- 2. Implement batch Q&A generation
- 3. Add training quality metrics
- 4. Enhanced device insights with more data

# **Testing Report**

■ Detailed Testing Report: /home/ubuntu/ai hub testing report.md

# 5. Sports Guide

# **Overview**

Simplified sports programming guide using The Rail Media API as the ONLY data source. All previous data sources (ESPN, TheSportsDB, Spectrum, etc.) have been removed for simplicity and maintainability.

**Version:** 4.0.0 - Simplified Implementation

**Last Updated:** October 16, 2025 **Data Source:** The Rail Media API ONLY

# **Key Changes (Version 4.0.0)**

# **Simplified Architecture**

• REMOVED: ESPN API integration

• REMOVED: TheSportsDB API integration

• REMOVED: Spectrum Channel Service

• REMOVED: Sunday Ticket Service

• REMOVED: Enhanced streaming sports service

• REMOVED: Mock data generation

• REMOVED: Multiple hardcoded channel lists

• KEPT: The Rail Media API as the ONLY data source

# **Benefits of Simplification**

- Single source of truth for all sports programming data
- Reduced code complexity (600+ lines → 300 lines)
- · Easier maintenance and debugging
- Consistent data format
- No API conflicts or data merging issues
- Comprehensive verbose logging for debugging

### **Features**

### **Core Functionality**

- Sports Programming Guide: Real-time sports TV guide data from The Rail Media
- Date Range Filtering: Query specific date ranges or number of days ahead
- Lineup Filtering: Filter by satellite, cable, or streaming lineup
- Search Functionality: Search for specific teams, leagues, or sports
- Comprehensive Logging: Verbose logging for all operations
- Ollama Integration: Al-powered query and analysis capabilities

### Supported Lineups

- SAT Satellite providers
- DRTV DirecTV
- DISH Dish Network
- CABLE Cable providers
- STREAM Streaming services

# **API Configuration**

Provider: The Rail Media

API Endpoint: https://guide.thedailyrail.com/api/v1

**User ID: 258351** 

API Key: Configured in .env file

#### **Environment Variables**

```
SPORTS_GUIDE_API_KEY=12548RK0000000d2bb701f55b82bfa192e680985919
SPORTS_GUIDE_USER_ID=258351
SPORTS_GUIDE_API_URL=https://guide.thedailyrail.com/api/v1
```

# **API Endpoints**

# POST /api/sports-guide

Fetch sports programming guide from The Rail Media API

# **Request Body:**

# Response:

```
"success": true,
 "requestId": "abc123",
 "dataSource": "The Rail Media API",
 "apiProvider": {
 "name": "The Rail Media",
 "url": "https://guide.thedailyrail.com/api/v1",
 "userId": "258351"
 },
 "fetchMethod": "fetchDateRangeGuide (7 days)",
 "data": {
 "listing_groups": [...]
 "statistics": {
 "totalListingGroups": 42,
 "totalListings": 156,
 "appliedFilters": [],
 "generatedAt": "2025-10-16T..."
 "filters": {
 "startDate": null,
 "endDate": null,
 "days": 7,
 "lineup": null,
 "search": null
 }
}
```

# **GET** /api/sports-guide

Get API information, status, and available endpoints

#### Response:

```
"success": true,
 "requestId": "xyz789",
 "version": "4.0.0",
 "name": "Simplified Sports Guide API",
 "description": "Sports programming guide using ONLY The Rail Media API",
 "dataSource": {
 "provider": "The Rail Media",
 "url": "https://guide.thedailyrail.com/api/v1",
 "userId": "258351",
 "apiKeySet": true,
 "configured": true
 },
 "endpoints": {...},
 "features": [...],
 "logging": {
 "enabled": true,
 "location": "PM2 logs (pm2 logs sports-bar-tv)",
 "format": "[timestamp] [Sports-Guide] LEVEL: message",
 "levels": ["INFO", "ERROR", "DEBUG"]
 },
 "supportedLineups": [...]
}
```

### **GET** /api/sports-guide?action=test-connection

Test The Rail Media API connection

# Response:

```
{
 "success": true,
 "requestId": "test123",
 "connectionTest": {
 "valid": true,
 "message": "API key is valid and working"
 },
 "timestamp": "2025-10-16T..."
}
```

# **GET** /api/sports-guide/status

Get current API configuration status

### Response:

```
{
 "success": true,
 "configured": true,
 "apiUrl": "https://guide.thedailyrail.com/api/v1",
 "userId": "258351",
 "apiKeySet": true,
 "apiKeyPreview": "12548RK0...5919"
}
```

# POST /api/sports-guide/verify-key

Verify API key validity

# **Request Body:**

```
{
 "apiKey": "your-api-key",
 "userId": "your-user-id"
}
```

# POST /api/sports-guide/update-key

Update API key (with validation)

# **Request Body:**

```
{
 "apiKey": "new-api-key",
 "userId": "new-user-id"
}
```

# **Ollama Al Integration**

The Sports Guide now includes comprehensive AI integration using Ollama for intelligent querying and analysis.

# **Ollama Configuration**

- Host: http://localhost:11434 (configurable via OLLAMA HOST)
- Model: phi3:mini (configurable via OLLAMA\_MODEL )
- Status: Active and operational

# **Ollama API Endpoints**

POST /api/sports-guide/ollama/query

Query Ollama about sports guide functionality

# **Default Query:**

```
{
 "query": "What sports games are on TV tonight?",
 "includeRecentLogs": true
}
```

# **Analyze Logs:**

```
{
 "action": "analyze-logs"
}
```

### **Get Recommendations:**

```
{
 "action": "get-recommendations",
 "userPreferences": {
 "favoriteTeams": ["Green Bay Packers", "Milwaukee Bucks"],
 "favoriteLeagues": ["NFL", "NBA"],
 "location": "Green Bay, Wisconsin"
 }
}
```

### **Test Connection:**

```
{
 "action": "test-connection"
}
```

# **GET** /api/sports-guide/ollama/query

Test Ollama connectivity

### Response:

```
{
 "success": true,
 "message": "Ollama is online and accessible",
 "model": "phi3:mini",
 "responseTime": 45
}
```

# **Ollama Features**

# 1. Intelligent Query Answering

- Natural language questions about sports programming
- Context-aware responses using recent logs
- Comprehensive system knowledge

#### 2. Log Analysis

- Automatic analysis of sports guide logs
- System health assessment
- Error detection and reporting
- Usage pattern identification

### 3. Personalized Recommendations

- Sports programming recommendations based on user preferences
- Location-based suggestions
- Team and league-specific recommendations

### 4. Debug Assistance

- Help troubleshooting issues

- Explain error messages
- Suggest solutions based on logs

# **Comprehensive Logging**

All sports guide operations are logged with comprehensive detail for debugging and monitoring.

# **Log Format**

```
[2025-10-16T12:34:56.789Z] [Sports-Guide] LEVEL: message
```

# **Log Levels**

- INFO: General information about operations
- ERROR: Error conditions and failures
- **DEBUG:** Detailed debugging information

# **Log Locations**

- PM2 Output Log: ~/.pm2/logs/sports-bar-tv-out.log
- PM2 Error Log: ~/.pm2/logs/sports-bar-tv-error.log

# **Viewing Logs**

### Real-time logs:

```
pm2 logs sports-bar-tv
```

# Filter for Sports Guide logs:

```
pm2 logs sports-bar-tv | grep "Sports-Guide"
```

### View specific log file:

```
tail -f ~/.pm2/logs/sports-bar-tv-out.log | grep "Sports-Guide"
```

### Search logs:

```
cat ~/.pm2/logs/sports-bar-tv-out.log | grep "Sports-Guide" | grep "ERROR"
```

# **Logged Operations**

- Request Processing: Every API request with unique request ID
- API Calls: The Rail API requests with parameters
- Data Fetching: Method used and response statistics
- Filtering: Applied filters and results
- Errors: Detailed error information with stack traces
- Statistics: Request counts, processing times, data volumes

# **Configuration Management**

# **Viewing Current Configuration**

1. Navigate to Sports Guide Configuration page

- 2. Click "API" tab
- 3. View current User ID and masked API Key
- 4. Check configuration status indicator

# **Updating Configuration**

#### Via UI:

- 1. Navigate to Sports Guide Configuration
- 2. Click "API" tab
- 3. Enter new User ID and API Key
- 4. Click "Verify API Key" to test
- 5. Click "Save Configuration"
- 6. Restart server for changes to take effect

### **Via Command Line:**

```
SSH into server
ssh -p 224 ubuntu@24.123.87.42

Edit .env file
nano /home/ubuntu/Sports-Bar-TV-Controller/.env

Update values:
SPORTS_GUIDE_API_KEY=your-new-key
SPORTS_GUIDE_USER_ID=your-new-user-id

Restart application
pm2 restart sports-bar-tv
```

# **Security**

- API Keys: Stored only in .env file (never in repository)
- **Key Masking:** UI shows only first 8 and last 4 characters
- Validation: API keys validated before saving
- Server-side Only: All API calls made from server, never client
- Environment Variables: Secure storage of sensitive credentials

# **Testing**

# **Test API Connection**

```
curl http://24.123.87.42:3000/api/sports-guide?action=test-connection
```

### Fetch Today's Guide

```
curl -X POST http://24.123.87.42:3000/api/sports-guide \
 -H "Content-Type: application/json" \
 -d '{}'
```

# Fetch 7-Day Guide

```
curl -X POST http://24.123.87.42:3000/api/sports-guide \
 -H "Content-Type: application/json" \
 -d '{"days": 7}'
```

### Search for NBA Games

```
curl -X POST http://24.123.87.42:3000/api/sports-guide \
 -H "Content-Type: application/json" \
 -d '{"search": "NBA", "days": 3}'
```

# Filter by DirecTV Lineup

```
curl -X POST http://24.123.87.42:3000/api/sports-guide \
 -H "Content-Type: application/json" \
 -d '{"lineup": "DRTV", "days": 1}'
```

#### **Test Ollama Connection**

```
curl http://24.123.87.42:3000/api/sports-guide/ollama/query
```

# **Query Ollama**

```
curl -X POST http://24.123.87.42:3000/api/sports-guide/ollama/query \
 -H "Content-Type: application/json" \
 -d '{"query": "What NFL games are on TV this week?"}'
```

### **Get AI Recommendations**

```
curl -X POST http://24.123.87.42:3000/api/sports-guide/ollama/query \
 -H "Content-Type: application/json" \
 -d '{
 "action": "get-recommendations",
 "userPreferences": {
 "favoriteTeams": ["Green Bay Packers"],
 "favoriteLeagues": ["NFL"]
 }
}'
```

# **Troubleshooting**

### Issue: "The Rail Media API not configured"

# Solution:

- 1. Check .env file has SPORTS GUIDE API KEY and SPORTS GUIDE USER ID
- 2. Verify values are correct
- 3. Restart application: pm2 restart sports-bar-tv

### Issue: "API key is invalid or unauthorized"

### Solution:

- 1. Verify API key is correct in .env file
- 2. Test API key using /api/sports-guide?action=test-connection
- 3. Contact The Rail Media support if key is correct but still failing

### Issue: No data returned

# **Solution:**

- 1. Check PM2 logs: pm2 logs sports-bar-tv | grep "Sports-Guide"
- 2. Verify date range is valid

- 3. Try fetching without filters first
- 4. Check The Rail Media API status

# **Issue: Ollama queries failing**

#### **Solution:**

- 1. Verify Ollama is running: curl http://localhost:11434/api/tags
- 2. Check Ollama model is downloaded: ollama list
- 3. Restart Ollama if needed: systemctl restart ollama (if using systemd)
- 4. Check logs for detailed error messages

# **Issue: Slow response times**

#### **Solution:**

- 1. Check network connectivity to The Rail Media API
- 2. Review logs for performance issues
- 3. Consider reducing date range for queries
- 4. Use Ollama to analyze logs for performance patterns

# **Migration Notes**

# **Upgrading from Version 3.x**

# What Changed:

- Removed all data sources except The Rail Media API
- Simplified API interface
- Added comprehensive logging
- Added Ollama AI integration
- Removed hardcoded channel lists

### **Migration Steps:**

- 1. Ensure The Rail Media API credentials are configured in .env
- 2. Update any frontend code that relied on old API response format
- 3. Test all sports guide functionality
- 4. Review logs to ensure proper operation
- 5. Update any custom integrations

#### **Breaking Changes:**

- Response format changed to focus on The Rail API data structure
- Removed mock data fallbacks
- Removed multi-source data merging
- Changed API response schema

### **Future Enhancements**

### **Planned Features:**

- Enhanced caching for frequently accessed data
- Webhook support for real-time updates
- User preference storage
- Advanced filtering options
- Integration with other system features (matrix routing, etc.)
- Mobile app support
- Push notifications for favorite teams

# Support

For issues with The Rail Media API:

- Website: https://guide.thedailyrail.com

- Support: Contact The Rail Media support team

For Sports Bar TV Controller issues:

- **Logs:** pm2 logs sports-bar-tv | grep "Sports-Guide"

- GitHub Issues: https://github.com/dfultonthebar/Sports-Bar-TV-Controller/issues

- Ollama Assistant: Use /api/sports-guide/ollama/query to ask questions

Last Updated: October 16, 2025

Version: 4.0.0 - Simplified Implementation Data Source: The Rail Media API Only

# 6. Streaming Platforms

### **Overview**

Management interface for streaming service accounts and configurations.

### **Features**

- · Platform account management
- Service configuration
- Integration settings

# 6.5. Global Cache IR Control

# **Overview**

Global Cache iTach IR Control system provides comprehensive infrared device management for cable boxes, AV receivers, and other IR-controlled equipment. The system supports both downloading IR codes from the Global Cache IR Database and **learning IR codes directly from physical remote controls**.

**Version:** 2.0 - With IR Learning Support **Last Updated:** October 17, 2025

**Status:** Production Ready

# **Key Features**

- Device Management: Add and manage Global Cache iTach devices (IP2IR, WF2IR, etc.)
- IR Code Database: Download pre-programmed IR codes from Global Cache cloud database
- IR Learning: Learn IR codes directly from physical remote controls
- Multi-Port Support: Configure multiple IR output ports per device
- Real-time Testing: Test device connectivity and IR transmission
- Comprehensive Logging: Verbose logging for debugging and monitoring

# **Global Cache Device Management**

# **Supported Models**

- iTach IP2IR: Ethernet to 3x IR outputs (Port 4998)
- iTach WF2IR: WiFi to 3x IR outputs (Port 4998)
- GC-100: Network adapter with IR/Serial/Relay support

# **Adding Devices**

- 1. Navigate to Device Configuration → Global Cache tab
- 2. Click "Add Device"
- 3. Enter device information:
  - **Device Name**: Friendly name (e.g., "Cable 1 iTach")
  - **IP Address**: Device network address (e.g., 192.168.5.110)
  - Port: TCP port (default 4998)
  - Model (optional): Device model identifier
- 4. Click "Add Device"
- 5. System will automatically test connectivity

# **Device Testing**

- Click the "Test" button on any device card
- System sends getdevices command to verify connectivity
- Results show device information and response time
- Status indicator shows online/offline state

# **IR Learning Feature**

**NEW:** The IR learning feature allows you to capture IR codes directly from physical remote controls without needing access to the Global Cache IR Database.

### **How IR Learning Works**

The Global Cache iTach devices have a built-in IR receiver (small hole near the power connector) that can capture IR signals from remote controls. The learning process:

- 1. Enable Learning Mode: Device enters learning mode via get IRL command
- 2. Capture Signal: Point remote at device and press button
- 3. Receive Code: Device sends captured IR code in Global Cache sendir format
- 4. Disable Learning: Automatic or manual via stop IRL command

### **Using IR Learning**

# **Step-by-Step Process:**

### 1. Navigate to IR Learning Tab

- Go to Device Configuration  $\rightarrow$  Global Cache
- Click the "IR Learning" tab

# 2. Select Device

- Choose a Global Cache device from the dropdown
- Device must be online and reachable

#### 3. Start Learning

- Click "Start Learning" button

- System sends get IRL command to device
- Wait for confirmation: "IR Learner Enabled"

### 4. Capture IR Signal

- Point your remote control at the Global Cache device
- Aim at the small IR receiver hole (near power connector)
- Press the button you want to learn
- Hold button for 1-2 seconds for best results

### 5. View Learned Code

- IR code appears automatically in text area
- Code is in Global Cache sendir format
- Example: sendir,1:1,1,38000,1,1,342,171,21,64,21,64,...

### 6. Save or Copy Code

- Option 1: Click "Copy" to copy code to clipboard
- Option 2: Enter a function name (e.g., "POWER") and click "Save to IR Device"
- Follow instructions to add code to an IR device

### **Important Notes:**

- Learning mode has a 60-second timeout
- Only one learning session at a time per device
- Device must not be configured for LED lighting
- Code is automatically stopped after learning completes
- Can manually stop learning with "Stop Learning" button

# **IR Learning API Endpoints**

# POST /api/globalcache/learn

Start IR learning session

```
{
 "deviceId": "clx123abc..."
}
```

### Response:

```
{
 "success": true,
 "status": "IR code learned successfully",
 "learnedCode": "sendir,1:1,1,38000,1,1,342,171,21,64,..."
}
```

# **DELETE** /api/globalcache/learn

Stop IR learning session

```
{
 "deviceId": "clx123abc..."
}
```

# **Response:**

```
{
 "success": true,
 "status": "IR Learner disabled"
}
```

# **IR Learning Commands (Global Cache API)**

### **Enable Learning:**

```
get_IRL\r
```

### Response:

```
IR Learner Enabled
```

### Learned Code (automatic):

```
sendir,1:1,1,38000,1,1,342,171,21,64,21,64,...\r
```

### **Disable Learning:**

```
stop_IRL\r
```

### Response:

```
IR Learner Disabled
```

# **Troubleshooting IR Learning**

### Problem: "IR Learner Unavailable"

- Cause: Device is configured for LED lighting control
- **Solution**: Reconfigure device to disable LED lighting mode
- Note: LED lighting and IR learning cannot be enabled simultaneously

# Problem: "Learning session timeout"

- Cause: No IR signal received within 60 seconds
- Solution:
- Ensure remote has fresh batteries
- Point remote directly at IR receiver hole
- Hold button for 1-2 seconds
- Try again with stronger IR signal

#### Problem: "Connection error"

- Cause: Cannot connect to Global Cache device
- Solution:
- Verify device is powered on
- Check network connectivity
- Test device in Device Management tab
- Verify IP address and port are correct

# Problem: "IR code not working after learning"

- Cause: Weak or incomplete IR signal captured
- Solution:
- Learn code again with remote closer to device
- Ensure remote batteries are fresh
- Try holding button longer (1-2 seconds)
- Verify learned code is not truncated

# **Comprehensive Logging**

All Global Cache operations include verbose logging for debugging and monitoring.

# **Log Format**

```
[GLOBAL CACHE] Starting IR learning
Device ID: clx123abc...
Timestamp: 2025-10-17T12:34:56.789Z
```

# **Logged Operations**

# **Device Management:**

- Device creation and deletion
- Connection testing
- Status updates

### IR Learning:

- Learning session start/stop
- Device connection attempts
- IR code capture events
- Learning timeouts and errors
- Code validation

# **Viewing Logs:**

```
Real-time logs
pm2 logs sports-bar-tv | grep "GLOBAL CACHE"

Search logs
cat ~/.pm2/logs/sports-bar-tv-out.log | grep "GLOBAL CACHE"

Error logs only
cat ~/.pm2/logs/sports-bar-tv-error.log | grep "GLOBAL CACHE"
```

# **Database Schema**

```
model GlobalCacheDevice {
 id String name String
 @id @default(cuid())
 id
 ipAddress String
 @unique
 port Int model String?
 @default(4998)
 status String
 @default("offline")
 lastSeen DateTime?
 ports GlobalCachePort[]
 createdAt DateTime
 @default(now())
 updatedAt DateTime
 @updatedAt
}
model GlobalCachePort {
 String
 @id @default(cuid())
 id
 deviceId
 String
 portNumber Int
portType String
assignedTo String?
 @default("IR")
 String?
 assigned {\tt DeviceId} \quad {\tt String?}
 irCodeSet String?
 @default(true)
 enabled
 Boolean
}
```

# Integration with IR Devices

Learned IR codes can be saved to IR devices for use in automated control:

- 1. **Create IR Device** (Device Configuration → IR Devices tab)
- 2. Add New Command to IR device
- 3. Paste Learned Code into IR Code field
- 4. Assign Function Name (e.g., "POWER", "CHANNEL UP")
- 5. Link to Global Cache Port for transmission
- 6. Test Command to verify functionality

# **Best Practices**

# **IR Learning**

- 1. Fresh Batteries: Use remote controls with fresh batteries
- 2. Close Proximity: Hold remote 2-6 inches from IR receiver
- 3. Direct Aim: Point directly at IR receiver hole
- 4. Button Hold: Hold button for 1-2 seconds for best capture
- 5. Verify Code: Test learned code immediately after capture
- 6. Document Codes: Add descriptive function names
- 7. Backup Codes: Keep copies of working codes

#### **Device Management**

- 1. Static IPs: Assign static IP addresses to Global Cache devices
- 2. Network Isolation: Place devices on same subnet as server
- 3. **Regular Testing**: Test device connectivity regularly
- 4. Firmware Updates: Keep device firmware up to date
- 5. Port Organization: Document which ports control which devices

# Limitations

#### 1. IR Learning:

- Cannot learn if device configured for LED lighting
- 60-second timeout per learning session
- One learning session at a time per device
- IR receiver location may vary by model

### 2. Device Support:

- Requires iTach firmware with IR learning support
- Not all Global Cache models support IR learning
- WiFi models may have network latency

### 3. IR Code Quality:

- Learned codes depend on original remote signal strength
- Some remotes use proprietary or non-standard IR protocols
- Complex codes may require multiple learning attempts

# **Future Enhancements**

#### **Planned Features:**

- -[] Bulk IR code learning with batch mode
- [ ] IR code library management and sharing
- [ ] Automatic IR device creation from learned codes
- [ ] IR code testing and verification tools
- [ ] Advanced code editing and manipulation
- [ ] Integration with IR device templates

# Support

For issues with Global Cache devices or IR learning:

### **Device Issues:**

- Test connectivity in Device Management tab
- Check PM2 logs for errors
- Verify network configuration
- Review Global Cache API documentation

### **IR Learning Issues:**

- Check logs for detailed error messages
- Verify device is not in LED lighting mode
- Test with different remote controls
- Contact Global Cache support for hardware issues

# **Documentation:**

- Global Cache iTach API: See global-cache-API-iTach.pdf
- IR Database API: See API-GlobalIRDB\_ver1.pdf
- System logs: pm2 logs sports-bar-tv

# 7. DirecTV Integration

# **Overview**

Integration with DirecTV receivers for sports bar TV control using the SHEF (Set-top Box HTTP Exported Functionality) protocol. The system allows adding, managing, and monitoring DirecTV receivers, retrieving device status and channel information, and routing them through the matrix switcher.

# **SHEF Protocol Information**

# SHEF (Set-top Box HTTP Exported Functionality)

Protocol Version: 1.12 (current H24/100 receiver)
 Documentation Version: 1.3.C (October 2011)

- **Port:** 8080 (default HTTP API port)

- Protocol: HTTP REST API- Response Format: JSON

### **Protocol Capabilities:**

- Device information (version, serial number, mode)
- Current channel and program information
- Remote control simulation (channel change, key presses)
- V Program guide data for specific channels
- V Device location information (multi-room setups)

#### **Protocol Limitations:**

- X NO subscription/package information
- X NO account details or billing data
- X NO entitled channels list
- X NO premium package status

# Why Subscription Data is Unavailable:

The SHEF API is designed for device control, not account management. Subscription data lives in DirecTV's cloud systems and would require integration with DirecTV's official business API, which is separate from the receiver's local HTTP API.

### **Current Status**

Last Updated: October 15, 2025, 7:08 PM
Overall Status: ✓ FULLY FUNCTIONAL

### **Working Features:**

- Receiver management and configuration
- V Device connectivity testing
- Real-time device status monitoring
- Current channel and program information
- V Device information display (receiver ID, access card, software version)
- Matrix switcher integration

### Fix Applied (October 15, 2025):

- Fixed subscription polling to correctly handle SHEF API limitations
- Removed incorrect logic that tried to parse API commands as subscription data
- Now displays real device information instead of attempting to fetch unavailable subscription data
- Shows receiver ID, access card ID, current channel, and program information

# **SHEF API Endpoints**

The DirecTV SHEF protocol provides the following HTTP endpoints on port 8080:

# **Device Information Endpoints**

### **GET** /info/getVersion

- Returns device version, receiver ID, access card ID, software version, and SHEF API version
- Example: http://192.168.5.121:8080/info/getVersion
- Response includes: receiverId, accessCardId, stbSoftwareVersion, version, systemTime

#### **GET** /info/getSerialNum

- Returns device serial number
- Example: http://192.168.5.121:8080/info/getSerialNum

#### **GET** /info/mode

- Returns device operational mode (0 = active, other values = standby/off)
- Example: http://192.168.5.121:8080/info/mode

### **GET** /info/getLocations

- Lists available client locations for multi-room setups
- Example: http://192.168.5.121:8080/info/getLocations

#### **GET** /info/getOptions

- Returns list of available API commands (NOT subscription data)
- This endpoint was previously misunderstood to provide subscription information
- Actually returns a list of API endpoints with their descriptions and parameters
- Example: http://192.168.5.121:8080/info/getOptions

# **TV Control Endpoints**

### **GET** /tv/getTuned

- Returns currently tuned channel and program information
- Example: http://192.168.5.121:8080/tv/getTuned
- Response includes: major, minor, callsign, title, programId, rating, etc.

# GET /tv/getProgInfo?major=<channel>&time=<timestamp>

- Returns program information for a specific channel at a given time
- Parameters: major (required), minor (optional), time (optional)
- Example: http://192.168.5.121:8080/tv/getProgInfo?major=202

#### **GET** /tv/tune?major=<channel>&minor=<subchannel>

- Tunes to a specific channel
- Parameters: major (required), minor (optional)
- Example: http://192.168.5.121:8080/tv/tune?major=202

### **Remote Control Endpoints**

### GET /remote/processKey?key=<keyname>

- Simulates pressing a remote control button
- Parameters: key (required) button name (e.g., "power", "menu", "chanup", "chandown")
- Example: http://192.168.5.121:8080/remote/processKey?key=power
- Available keys: power, poweron, poweroff, format, pause, rew, replay, stop, advance, ffwd, record, play, guide, active, list, exit, back, menu, info, up, down, left, right, select, red, green, yellow, blue, chanup, chandown, prev, 0-9, dash, enter

#### GET /serial/processCommand?cmd=<hex command>

- Sends a raw serial command to the receiver (advanced users only)
- Parameters: cmd (required) hexadecimal command string

# **Deprecated Endpoints (Do Not Use)**

**GET** /dvr/getPlaylist - Deprecated in SHEF v1.3.C

GET /dvr/play - Deprecated in SHEF v1.3.C

### **Features**

# **Receiver Management**

- Add DirecTV Receivers: Configure receivers with IP address, port, and receiver type
- Matrix Integration: Assign receivers to specific matrix input channels (1-32)
- Connection Testing: Test connectivity to DirecTV receivers
- Subscription Data: Retrieve active subscriptions and sports packages
- Status Monitoring: Real-time connection status indicators

# **Receiver Configuration**

- Device Name: Custom label for identification
- IP Address: Network address of DirecTV receiver
- Port: Default 8080 (DirecTV API port)
- Receiver Type: Genie HD DVR, HR24, etc.
- Matrix Input Channel: SELECT dropdown with 32 input channels
- Format: "Input 1: Cable Box 1 (Cable Box)"
- Links receiver to specific matrix input for routing

# Testing Results (October 15, 2025)

# **✓** Successful Operations

#### 1. Receiver Creation (PASSED)

- Successfully created DirecTV receivers with full configuration
- Matrix Input Channel field is functional as SELECT dropdown
- All 32 matrix input channels available in dropdown
- Receiver appears in UI with proper configuration
- Status indicator shows "Connected" (green checkmark)

#### 2. Receiver Deletion (PASSED)

- Successfully removed multiple receivers (tested with 9 receivers)
- Deletion confirmation dialog appears for each receiver
- Each deletion processed successfully
- UI updates correctly after each deletion

### 3. Form Validation (PASSED)

- IP address validation working correctly
- Port number validation (default 8080)
- Matrix input channel selection functional
- All form fields properly integrated with React state

# X Failed Operations & Known Issues

### 1. Subscription Data Retrieval (FAILED)

- **Status:** X FAILS when no physical receiver present

- Error Message: "Polling Failed Unable to connect to DirecTV receiver"
- Dialog Display:
- Title: "Device Subscriptions [Receiver Name]"
- Error Badge: Red "Error" indicator
- Error Message: "Polling Failed Unable to connect to DirecTV receiver"
- Active Subscriptions: 0Sports Packages: 0
- Last Updated: Timestamp

#### 2. Connection Test Results

- Visual Indicator: Shows "Connected" (green) in UI
- Actual Status: Cannot verify without physical hardware
- Limitation: UI may show connected even when receiver is unreachable

# **Error Messages & Diagnostics**

# **Subscription Polling Error**

```
Error: Unable to connect to DirecTV receiver
Status: Polling Failed
Active Subscriptions: 0
Sports Packages: 0
Timestamp: [Date/Time of polling attempt]
```

#### **Root Causes:**

- 1. No Physical Device: IP address has no actual DirecTV receiver
- 2. **Network Connectivity:** Receiver unreachable from server network
- 3. Receiver Offline: Device powered off or disconnected
- 4. Firewall/Port Blocking: Port 8080 blocked by network firewall
- 5. API Endpoint Issue: Backend API connection problems

### Form Input Handling Issues

During testing, direct typing in React form fields did not update state properly. Workaround implemented using native JavaScript:

```
const nativeInputValueSetter = Object.getOwnPropertyDescriptor(
 window.HTMLInputElement.prototype, "value"
).set;
nativeInputValueSetter.call(inputElement, 'value');
inputElement.dispatchEvent(new Event('input', { bubbles: true }));
```

# **Verbose Logging Implementation**

The DirecTV system includes comprehensive logging for debugging and monitoring:

### Log Locations

- PM2 Logs: pm2 logs sports-bar-tvLog Files: /home/ubuntu/.pm2/logs/
- sports-bar-tv-out.log Standard output
- sports-bar-tv-error.log Error output

# **Logged Operations**

#### **Receiver Creation:**

```
[DirecTV] Creating new receiver: Test DirecTV

[DirecTV] IP: 192.168.5.121, Port: 8080

[DirecTV] Matrix Channel: 1 (Input 1: Cable Box 1)

[DirecTV] Receiver created successfully
```

### **Connection Testing:**

```
[DirecTV] Testing connection to 192.168.5.121[8080]
[DirecTV] Connection attempt: [SUCCESS/FAILED]
[DirecTV] Response time[X] [X]ms
```

#### **Subscription Polling:**

```
[DirecTV] Polling subscriptions for receiver: Test DirecTV
[DirecTV] API endpoint: http://192.168.5.121:8080/api/subscriptions
[DirecTV] ERROR: Unable to connect to DirecTV receiver
[DirecTV] Error details: [Connection timeout/Network unreachable/etc.]
```

#### **Receiver Deletion:**

```
[DirecTV] Deleting receiver: Test DirecTV (ID: xxx)
[DirecTV] Receiver deleted successfully
```

# **Accessing Logs**

#### **View Real-time Logs:**

```
pm2 logs sports-bar-tv
```

### **View Specific Log File:**

```
tail -f ~/.pm2/logs/sports-bar-tv-out.log
tail -f ~/.pm2/logs/sports-bar-tv-error.log
```

#### **Search Logs for DirecTV Events:**

```
pm2 logs sports-bar-tv | grep DirecTV
cat ~/.pm2/logs/sports-bar-tv-out.log | grep "DirecTV"
```

# **UI Components & Behavior**

#### **Receiver Card Interface**

When a DirecTV receiver is selected, three action buttons appear:

- 1. Purple Button (Leftmost): Retrieve subscription data
- 2. Blue Button (Middle): Additional functionality (TBD)
- 3. Red/Orange Button (Rightmost): Delete receiver

### **Status Indicators**

• Green Checkmark: "Connected" status

- Red Badge: Error or disconnected status
- Loading Spinner: Operation in progress

# **Matrix Input Channel Field**

- Type: SELECT dropdown (not text input)
- Position: Second select element in form
- Options: 32 channels with descriptive labels
- Value Format: String numbers "1" through "32"
- Label Format: "Input [N]: [Label] ([Type])"

# **API Endpoints**

# POST /api/directv/receivers

Create a new DirecTV receiver configuration.

### **Request Body:**

```
"deviceName": "Test DirecTV",
"ipAddress": "192.168.5.121",
"port": 8080,
"receiverType": "Genie HD DVR",
"matrixInputChannel": 1
}
```

#### Response:

```
"success": true,
"receiver": {
 "id": "xxx",
 "deviceName": "Test DirecTV",
 "ipAddress": "192.168.5.121",
 "port": 8080,
 "receiverType": "Genie HD DVR",
 "matrixInputChannel": 1,
 "connected": true,
 "createdAt": "2025-10-15T18:10:00.000Z"
}
```

# **GET** /api/directv/receivers

Retrieve all configured DirecTV receivers.

# **DELETE** /api/directv/receivers/[id]

Delete a specific DirecTV receiver.

#### **POST** /api/directv/test-connection

Test connection to a DirecTV receiver.

# **Request Body:**

```
{
 "receiverId": "xxx"
}
```

# Response:

```
{
 "success": true,
 "connected": true,
 "responseTime": 45
}
```

# POST /api/directv/subscriptions

Retrieve subscription data from DirecTV receiver.

# **Request Body:**

```
{
 "receiverId": "xxx"
}
```

# **Response (Success):**

### Response (Error):

```
"success": false,
"error": "Unable to connect to DirecTV receiver",
"activeSubscriptions": 0,
"sportsPackages": 0,
"lastUpdated": "2025-10-15T18:10:26.000Z"
}
```

# **Database Schema**

```
model DirecTVReceiver {
 @id @default(cuid())
 id
 String
 deviceName
 String
 ipAddress
 String
 port
 Int
 @default(8080)
 receiverType
 String
 matrixInputChannel Int
 Boolean @default(false)
 connected
 connected Boolean lastConnected DateTime?
 activeSubscriptions Int?
 sportsPackages
 Int?
 lastPolled
 DateTime?
 createdAt
 DateTime @default(now())
 updatedAt
 DateTime @updatedAt
 @@unique([ipAddress, port])
}
```

# **Known Issues & Limitations**

# 1. Physical Hardware Required

**Issue:** Subscription polling and advanced features require actual DirecTV hardware **Impact:** Cannot fully test or use subscription features without physical receiver **Workaround:** UI and management features work independently of hardware

Status: EXPECTED BEHAVIOR - Not a bug

# 2. Connection Status Ambiguity

Issue: UI may show "Connected" status even when receiver is unreachable

Impact: Users may be misled about actual device connectivity

Recommendation: Implement periodic health checks and more accurate status reporting

**Priority: MEDIUM** 

### 3. Form Input React State Sync

**Issue:** Direct typing in form fields may not update React state **Impact:** Values may not save properly on form submission

Workaround: Use native JavaScript input setter with event dispatch

Status: Workaround implemented, consider fixing React state management

**Priority: LOW** 

# 4. Network Topology Dependency

Issue: Server must be on same network as DirecTV receivers

Impact: Cannot manage receivers on different VLANs/subnets without routing

Recommendation: Document network requirements, consider VPN/tunnel for remote access

**Priority: MEDIUM** 

# **Troubleshooting Guide**

Problem: "Unable to connect to DirecTV receiver"

**Diagnostic Steps:** 

# 1. Verify Network Connectivity

```bash

```
# SSH into server
ssh -p 224 ubuntu@24.123.87.42
# Test ping to receiver
ping 192.168.5.121
# Test HTTP connectivity
curl http://192.168.5.121:8080
```

1. Check Receiver Status

- Verify DirecTV receiver is powered on
- Confirm receiver is connected to network
- Check receiver's IP address in network settings
- Verify receiver's network LED indicator

2. Validate Configuration

- Confirm IP address is correct in UI
- Verify port number (should be 8080)
- Check receiver is configured for network control
- Ensure receiver firmware is up to date

3. Review Backend Logs

```
"`bash
# Check PM2 logs for DirecTV errors
pm2 logs sports-bar-tv | grep DirecTV
```

```
# Check last 50 lines of error log tail -50 ~/.pm2/logs/sports-bar-tv-error.log
```

1. Test Firewall/Port Access

```
"> bash
# Test if port 8080 is accessible
telnet 192.168.5.121 8080

# Or use nc (netcat)
nc -zv 192.168.5.121 8080
```

1. Verify Network Routing

```
"" bash

# Check routing table
route -n

# Trace route to receiver
traceroute 192.168.5.121
```

Problem: Receiver shows "Connected" but subscription data fails

Possible Causes:

- Connection test endpoint responds but subscription API doesn't
- Receiver authentication required for subscription data

- API endpoint path incorrect for receiver model
- Receiver doesn't support network subscription queries

Solutions:

- 1. Review DirecTV receiver's network API documentation
- 2. Check if authentication/credentials required
- 3. Verify API endpoint paths for specific receiver model
- 4. Test with DirecTV's official API testing tools

Problem: Form submission not saving values

Solution:

- 1. Clear browser cache and reload page
- 2. Check browser console for JavaScript errors
- 3. Verify React state updates in browser DevTools
- 4. Use workaround with native input setters if needed

Problem: Matrix input channel not routing correctly

Diagnostic Steps:

- 1. Verify matrix input channel number is correct (1-32)
- 2. Check matrix switcher configuration in System Admin
- 3. Test matrix switching directly without DirecTV
- 4. Verify input channel is properly configured in matrix

Recommendations for Production Use

Network Configuration

- 1. Isolated VLAN: Place DirecTV receivers on dedicated VLAN
- 2. Static IPs: Assign static IP addresses to all receivers
- 3. **DNS Records:** Create DNS entries for receivers (e.g., directv-1.local)
- 4. Port Forwarding: Configure if receivers are on different subnet

Monitoring & Maintenance

- 1. **Health Checks:** Implement periodic connection health checks (every 5 minutes)
- 2. Status Alerts: Send notifications when receivers go offline
- 3. Log Rotation: Ensure PM2 logs don't fill disk space
- 4. Backup Configuration: Backup receiver configurations daily

Testing with Real Hardware

To properly test and use DirecTV features:

1. Acquire Compatible Receiver:

- Genie HD DVR (HR44, HR54)
- HR24 HD DVR
- Or other network-enabled DirecTV receivers

2. Network Setup:

- Connect receiver to network
- Assign static IP or create DHCP reservation
- Verify network connectivity from server

3. Receiver Configuration:

- Enable network control in receiver settings

- Configure IP address and port
- Test receiver's web interface directly

4. Application Testing:

- Add receiver with correct IP and settings
- Test connection functionality
- Verify subscription polling works
- Test matrix routing integration

Security Considerations

- 1. API Access: Secure DirecTV API endpoints if exposed
- 2. Network Segmentation: Isolate receivers from guest networks
- 3. Access Control: Implement authentication for receiver management
- 4. Audit Logging: Log all receiver configuration changes

Integration with Matrix Switcher

DirecTV receivers integrate seamlessly with the Wolfpack HDMI matrix:

- 1. **Configuration:** Assign receiver to specific matrix input channel
- 2. Routing: Route receiver to any TV output via matrix control
- 3. Status: Monitor receiver status alongside matrix outputs
- 4. Control: Manage receiver and routing from single interface

Example Workflow:

- 1. Add DirecTV receiver on matrix input channel 5
- 2. Configure receiver as "Sports Bar DirecTV Main"
- 3. Route to TV outputs as needed for sports events
- 4. Monitor connection status and subscriptions
- 5. Verify sports packages include desired channels

Future Enhancements

Planned Features:

- [] Implement periodic health checks with accurate status reporting
- [] Add receiver channel control (change channels remotely)
- [] Integrate with Sports Guide for auto-tuning
- [] Support multiple receiver types (clients, mini-Genies)
- [] Implement receiver discovery on network
- [] Add bulk receiver management
- [] Create receiver groups for simultaneous control
- [] Implement receiver event scheduling

Under Consideration:

- Remote recording management
- DVR playlist integration
- Channel favorites sync
- Multi-receiver coordination
- Advanced diagnostic tools

8. IR Device Setup & Global Cache Integration

Overview

Version: 1.0

Last Updated: October 17, 2025

Status: Production Ready

The IR Device Setup system provides comprehensive management of infrared-controlled devices (cable boxes, satellite receivers, AV receivers, etc.) through Global Cache iTach IR control devices. This unified system allows users to configure IR devices, assign them to Global Cache devices and ports, and control them remotely.

System Architecture

Components

1. Global Cache iTach Devices

- Network-connected IR blasters
- Support 3 IR ports per device
- TCP communication on port 4998
- Status monitoring and health checks

2. IR Devices

- Physical devices to be controlled (cable boxes, receivers, etc.)
- Each device linked to a specific Global Cache device and port
- Matrix switcher integration for video routing
- IR command database integration

3. IR Command Database

- Global Cache online IR code database
- Thousands of device codesets
- Search by brand/model
- Automatic command download

Features

Global Cache Device Management

- Add/Remove Devices: Configure Global Cache iTach devices with IP address and port
- Connection Testing: Test connectivity to devices
- Port Configuration: Manage 3 IR ports per device
- Status Monitoring: Real-time online/offline status
- Port Assignment: Track which IR device is assigned to each port

IR Device Configuration

- Device Creation: Add IR devices with:
- Name (e.g., "Cable Box 1")
- Type (Cable Box, Satellite, AV Receiver, etc.)
- Brand (e.g., "DirecTV", "Dish", "Denon")
- Model (optional)
- Global Cache device selection (dropdown)
- Port number selection (dropdown, filtered by device)
- Matrix input channel (optional)

- Description (optional)
- Edit Functionality: Modify existing IR device configurations
- Delete Functionality: Remove IR devices with confirmation
- IR Database Integration: Search and download IR codes from Global Cache database
- Command Management: View and organize IR commands per device

Verbose Logging

All operations include comprehensive console logging:

- Component mounting and initialization
- Device loading and counts
- Global Cache device selection
- Port selection changes
- Add/Update/Delete operations
- API calls and responses
- Error handling with detailed messages

API Endpoints

Global Cache Devices

GET /api/globalcache/devices

- List all Global Cache devices with ports
- Returns device status, IP address, port info

POST /api/globalcache/devices

- Add new Global Cache device
- Tests connection during creation
- Creates 3 IR ports automatically

DELETE /api/globalcache/devices/[id]

- Remove Global Cache device
- Cascades to delete port assignments

IR Devices

GET /api/ir/devices

- List all IR devices
- Includes ports, commands, and Global Cache assignments
- Comprehensive logging

POST /api/ir/devices

- Create new IR device
- Accepts globalCacheDeviceId and globalCachePortNumber
- Validates required fields (name, deviceType, brand)
- Logs all operations

PUT /api/ir/devices/[id]

- Update existing IR device
- Supports partial updates
- Logs changes with before/after values

DELETE /api/ir/devices/[id]

- Remove IR device

- Deletes associated commands
- Requires confirmation

IR Commands

GET /api/ir/database/brands

- Search IR database for device brands

GET /api/ir/database/models

- Get models for specific brand

POST /api/ir/database/download

- Download IR commands for device
- Saves to database linked to IR device

Database Schema

GlobalCacheDevice

```
model GlobalCacheDevice {
 id
             String
                                 @id @default(cuid())
 name
             String
 ipAddress String
                                 @unique
                                 @default(4998)
            Int
 port
 model
             String?
  status
                                 @default("offline")
             String
  lastSeen
             DateTime?
             GlobalCachePort[]
 ports
 createdAt
             DateTime
                                 @default(now())
 updatedAt
             DateTime
                                 @updatedAt
}
```

GlobalCachePort

```
model GlobalCachePort {
 id
                                      @id @default(cuid())
                    String
  deviceId
                    String
  device
                    GlobalCacheDevice @relation(...)
  portNumber
                    Int
  portType
                    String
                                      @default("IR")
  assignedTo
                    String?
  assignedDeviceId String?
                                      @default(true)
  enabled
                    Boolean
  irDevice
                    IRDevice?
                                      @relation(...)
  createdAt
                    DateTime
                                      @default(now())
  updatedAt
                    DateTime
                                      @updatedAt
}
```

IRDevice

```
model IRDevice {
                                            @id @default(cuid())
  id
                        String
  name
                        String
  deviceType
                        String
  brand
                        String
  model
                        String?
                        Int?
  matrixInput
  matrixInputLabel
                        String?
  irCodeSetId
                        String?
  globalCacheDeviceId String?
                                            // NEW FIELD
  globalCachePortNumber Int?
                                            // NEW FIELD
  description
                        String?
  status
                        String
                                            @default("active")
                        GlobalCachePort[]
  ports
                        IRCommand[]
  commands
  createdAt
                        DateTime
                                            @default(now())
                        DateTime
                                            @updatedAt
  updatedAt
}
```

IRCommand

```
model IRCommand {
                             @id @default(cuid())
 id
                    String
  deviceId
                   String
                   IRDevice @relation(...)
  device
  functionName
                   String
  irCode
                   String
                   String?
  category
  description
                   String?
  createdAt
                   DateTime @default(now())
  updatedAt
                   DateTime @updatedAt
}
```

User Interface

Device Configuration Page (/device-config)

- Two Tabs:
 - 1. Global Cache: Manage Global Cache iTach devices
 - 2. IR Devices: Configure IR-controlled devices

IR Device Setup Interface

Add/Edit Form Fields:

- Device Name (required)
- Device Type (dropdown: Cable Box, Satellite, AV Receiver, etc.)
- Brand (required)
- Model (optional)
- Global Cache Device (dropdown with device name, IP, status)
- Port Number* (dropdown, shows Port 1-3, filtered by selected device)
- Matrix Input Channel (optional)
- Matrix Input Label (optional)
- Description (optional)

Device Cards Display:

- Device name, brand, model, type

- Command count badge
- Global Cache device name
- Global Cache port number
- Matrix input information
- Codeset ID
- Description
- Available commands (first 10 shown)
- Action buttons:
- IR Database (search and download codes)
- Edit (modify configuration)
- Delete (remove device)

Workflow

Adding a New IR Device

- 1. Navigate to Device Configuration (/device-config)
- 2. Switch to IR Devices Tab
- 3. Click "Add IR Device"
- 4. Fill in device information:
 - Name: "Cable Box 1"
 - Type: "Cable Box"
 - Brand: "Spectrum"
 - Model: "DCX3600" (optional)
- 5. Select Global Cache Device:
 - Choose from dropdown: "GC iTach 1 (192.168.5.110) online"
 - Port dropdown activates
- 6. Select Port Number:
 - Choose "Port 1 (IR)" or "Port 2 (IR)" or "Port 3 (IR)"
 - Shows if port is already assigned
- 7. Optionally set Matrix Input:
 - Channel: 5
 - Label: "Cable"
- 8. Add Description (optional)
- 9. Click "Add Device"
- 10. Device appears in list with all configured details

Downloading IR Codes

- 1. Click "IR Database" button on device card
- 2. Search for device by brand and model
- 3. Select codeset from results
- 4. Download commands to device
- 5. Commands appear in device card

Editing an IR Device

- 1. Click "Edit" button on device card
- 2. Form pre-fills with current values
- 3. Modify any fields:
 - Change Global Cache device

- Switch port number
- Update matrix channel
- 4. Click "Update Device"
- 5. Device updates immediately

Console Logging Examples

Component Mount

```
[IR DEVICE SETUP] Component mounted Timestamp: 2025-10-17T10:30:00.000Z
```

Loading Devices

```
[ [IR DEVICE SETUP] Loading IR devices...

[ [IR DEVICE SETUP] IR devices loaded: 3
[ [IR DEVICE SETUP] Loading Global Cache devices...

[ [IR DEVICE SETUP] Global Cache devices loaded: 2
```

Adding Device

API Response

```
Image: IR DEVICES] Creating new IR device
    Timestamp: 2025-10-17T10:30:00.000Z

Name: Cable Box 1
    Type: Cable Box
    Brand: Spectrum
    Model: DCX3600
    Global Cache Device: cm1234567890
    Global Cache Port: 1

Image: IR DEVICES] Device created successfully
    ID: clm9876543210
```

Troubleshooting

Issue: No Global Cache devices in dropdown

Solution:

- 1. Navigate to Global Cache tab
- 2. Add Global Cache iTach device with IP address
- 3. Test connection

- 4. Return to IR Devices tab
- 5. Refresh page if needed

Issue: Port dropdown is disabled

Solution:

- 1. Select a Global Cache device first
- 2. Port dropdown will activate automatically
- 3. Ports are filtered to show only enabled ports on selected device

Issue: Device not controlling IR equipment

Diagnostic Steps:

- 1. Verify Global Cache device is online (check status in Global Cache tab)
- 2. Confirm correct port number is selected
- 3. Test Global Cache connection manually
- 4. Ensure IR emitter is connected to correct port
- 5. Download IR codes from database if not done
- 6. Test IR commands through API

Issue: IR codes not working

Solution:

- 1. Try downloading different codeset from IR database
- 2. Verify IR emitter is positioned correctly near device IR sensor
- 3. Check for IR interference from other sources
- 4. Test Global Cache device with other commands

Issue: "Argument irCode is missing" error when downloading IR codes

Error Message:

```
Error downloading VOLUME UP:
Invalid `prisma.iRCommand.create()` invocation:

data: {
    deviceId: "...",
    functionName: "VOLUME UP",
    hexCode: undefined,
    codeSetId: "6935",
    category: "Volume",
+ irCode: String

Argument `irCode` is missing.

Argument `irCode` is missing.
```

Root Causes:

- 1. API Rate Limit: Global Cache API has daily download limits
- 2. Invalid API Key: API key expired or not logged in
- 3. Missing Code Data: API returned error response instead of code data
- 4. Network Issues: Connection problems with Global Cache servers

Fix Applied (October 17, 2025):

The system now includes comprehensive validation and error handling:

1. API Response Validation: Checks if response is a CodeResponse (error) vs Code (success)

- 2. Error Code Mapping: Maps Global Cache API error codes to human-readable messages:
 - Code 2: API Key not found
 - Code 3: User not logged in
 - Code 4: Too many requests (rate limit exceeded)
 - Code 5: Unknown output type
 - Code 6: Direct output not allowed for account
 - Code 7: API key required but not provided
 - Code 8: Email send failed
 - Code 9: Unknown format requested
- 3. Missing Field Detection: Validates that Code1 field exists before creating database entry
- 4. Enhanced Logging: Detailed console logs show raw API responses and validation steps

Diagnostic Steps:

1. Check PM2 logs for detailed error messages:

pm2 logs sports-bar-tv | grep "IR DATABASE"

1. Look for API error codes in logs:

[IR DATABASE] Received CodeResponse (not Code)

Status: failure

Message: Too many IR codes already requested today

Error Code: 4

2. Verify API credentials:

- Check if logged in to Global Cache IR Database
- Verify API key is valid and active
- Re-login if necessary

3. Check rate limits:

- Global Cache limits downloads per day
- Wait 24 hours if rate limit exceeded
- Consider using sandbox mode for testing

Solutions:

1. Rate Limit Exceeded (Error Code 4):

- Wait until next day (resets at midnight UTC)
- Use sandbox mode for testing: Add sandbox=true to API URL
- Prioritize essential commands only

2. API Key Issues (Error Code 2, 3, 7):

- Navigate to IR Device Setup
- Click "IR Database Login"
- Enter credentials and re-login
- Verify API key is stored in database

3. Network/Connection Issues:

- Check internet connectivity
- Verify firewall allows outbound HTTPS to irdb.globalcache.com:8081
- Test connection to Global Cache API

4. Invalid Code Data:

- Try different codeset for same device

- Report issue to Global Cache support
- Use learning mode if available

Verification:

After fix, successful downloads show:

[IR DATABASE] Code downloaded successfully
Function: VOLUME UP

Code1 length: 68 HexCode1 length: 32

✓ Created command: VOLUME UP

Prevention:

- Monitor daily download count
- Download codes in batches to avoid rate limits
- Keep API credentials current and valid
- Use comprehensive error handling in custom integrations

Best Practices

Global Cache Device Naming

- Use descriptive names: "GC iTach 1 Bar Area"
- Include location for easy identification
- Document which area/rack device is located in

Port Assignment

- Document port assignments in device descriptions
- Use consistent naming conventions
- Track cable connections physically

IR Device Organization

- · Use clear, consistent naming
- Include location in name: "Cable Box 1 TV 3"
- Set matrix input for automatic video routing
- Add detailed descriptions for complex setups

Testing

- Test connection after adding Global Cache device
- Verify IR codes work before production use
- · Test edit functionality periodically
- Monitor logs for errors

Integration with Other Systems

Matrix Switcher

- IR devices can specify matrix input channel
- Enables automatic video routing
- · Coordinates IR control with video switching

Bartender Remote

• IR commands can be triggered from remote interface

- · Simplifies bartender operations
- · Reduces complexity for staff

Schedule System

- IR devices can be controlled via scheduled events
- Automatic channel changes
- Power on/off automation

Future Enhancements

Planned Features:

- [] Macro commands (multiple IR commands in sequence)
- [] IR learning mode (learn from physical remote)
- -[] Bulk IR code downloads
- [] Command testing interface
- [] Activity monitoring and logging
- [] Port conflict detection
- [] Automatic port assignment suggestions
- [] IR code validation and testing

Migration Notes

October 17, 2025 Update:

- Added globalCacheDeviceId field to IRDevice model
- Added globalCachePortNumber field to IRDevice model
- Updated API routes to handle new fields
- Added dropdown selectors in UI
- Implemented edit functionality
- Added comprehensive verbose logging throughout

Database Migration:

```
cd /home/ubuntu/Sports-Bar-TV-Controller
npx prisma db push
npx prisma generate
```

Component Updates:

- IRDeviceSetup.tsx: Added Global Cache device/port selection
- API routes: Added logging and new field handling
- Database schema: Added new fields with indexes

Support

Viewing Logs:

```
# Real-time logs
pm2 logs sports-bar-tv

# Search for IR device operations
pm2 logs sports-bar-tv | grep "IR DEVICE SETUP"

# Search for API operations
pm2 logs sports-bar-tv | grep "IR DEVICES"
```

Testing API:

```
# List IR devices
curl http://24.123.87.42:3000/api/ir/devices

# List Global Cache devices
curl http://24.123.87.42:3000/api/globalcache/devices

# Test Global Cache connection
curl -X POST http://24.123.87.42:3000/api/globalcache/devices/[id]/test
```

9. Remote Control

Overview

Bartender Remote interface for quick TV and audio control.

Features

- · Quick TV source selection
- Matrix status display
- Bar layout visualization
- Input source shortcuts

10. System Admin

Overview

Administrative tools for system management, testing, and maintenance.

Features

Wolfpack Configuration

- Matrix IP address setup
- · Connection testing
- Switching tests

Matrix Inputs/Outputs

- Input/output labeling
- Enable/disable configuration
- Schedule participation settings

System Logs

- Application logs
- Error tracking
- · Activity monitoring

Backup Management

- · Manual backup creation
- · Backup restoration

Automated backup status

TODO Management

- · Task tracking
- Priority management
- Status updates

Wolfpack Integration

POST /api/wolfpack/test-connection

Test connectivity to Wolfpack matrix:

```
"ipAddress": "192.168.1.100"
}
```

POST /api/wolfpack/test-switching

Test matrix switching functionality

Database Schema

```
model WolfpackConfig {
 id
        Int
                  @id @default(autoincrement())
 ipAddress String
                  @unique
 name String?
 createdAt DateTime @default(now())
 updatedAt DateTime @updatedAt
}
```

TODO Management

The TODO management system provides task tracking and project management capabilities. The system automatically maintains a TODO LIST.md file that reflects the current state of all tasks in the database.



Important: TODO_LIST.md is Auto-Generated

DO NOT EDIT TODO_LIST.md MANUALLY

The TODO_LIST.md file is automatically generated and updated by the TODO management system. Any manual changes will be overwritten when the system syncs. Always use the TODO API to add, update, or delete tasks.

The auto-generation happens:

- When a TODO is created via the API
- When a TODO is updated via the API
- When a TODO is deleted via the API
- During periodic system syncs

Database Schema

```
model Todo {
                  String
                                @id @default(cuid())
 id
  title
                  String
 description
                  String?
                                @default("MEDIUM") // "LOW", "MEDIUM", "HIGH", "CRIT-
                  String
 priority
ICAL"
                                @default("PLANNED") // "PLANNED", "IN PROGRESS", "TEST
                  String
  status
ING", "COMPLETE"
 category
                 String?
 tags
                  String?
                                // JSON array of tags
 createdAt
                 DateTime
                                @default(now())
                 DateTime
                                @updatedAt
 updatedAt
                 DateTime?
 completedAt
 documents
                 TodoDocument[]
}
model TodoDocument {
                  String
                         @id @default(cuid())
 id
 todoId
                  String
 filename
                  String
  filepath
                  String
 filesize
                  Int?
  mimetype
                  String?
 uploadedAt
                  DateTime @default(now())
                           @relation(fields: [todoId], references: [id], onDelete: Cas
 todo
                 Todo
cade)
 @@index([todoId])
}
```

API Endpoints

GET /api/todos - List all TODOs

Retrieve all TODOs with optional filtering.

Query Parameters:

```
- status (optional) - Filter by status: PLANNED , IN_PROGRESS , TESTING , COMPLETE
```

- priority (optional) Filter by priority: LOW , MEDIUM , HIGH , CRITICAL
- category (optional) Filter by category string

Response:

```
"success": true,
  "data": [
     "id": "cmgki7fkg0001vsfg6ghz142f",
      "title": "Fix critical bug",
      "description": "Detailed description...",
      "priority": "CRITICAL",
      "status": "PLANNED",
      "category": "Bug Fix",
      "tags": "[\"ai-hub\", \"database\"]",
      "createdAt": "2025-10-10T07:07:10.000Z",
      "updatedAt": "2025-10-10T07:07:10.000Z",
      "completedAt": null,
      "documents": []
    }
 ]
}
```

Example cURL:

```
# Get all TODOs
curl http://localhost:3000/api/todos

# Get only high priority TODOs
curl http://localhost:3000/api/todos?priority=HIGH

# Get in-progress tasks
curl http://localhost:3000/api/todos?status=IN_PROGRESS
```

POST /api/todos - Create new TODO

Add a new TODO item to the system. The TODO LIST.md file will be automatically updated.

Request Body:

```
"title": "Task title (required)",
  "description": "Detailed task description (optional)",
  "priority": "MEDIUM",
  "status": "PLANNED",
  "category": "Category name (optional)",
  "tags": ["tag1", "tag2"]
}
```

Priority Levels:

- LOW Minor tasks, nice-to-have features
- MEDIUM Standard priority (default)
- HIGH Important tasks requiring attention
- CRITICAL Urgent tasks blocking functionality

Status Values:

- PLANNED Task is planned but not started (default)
- IN_PROGRESS Currently being worked on
- TESTING Implementation complete, being tested
- COMPLETE Task finished and verified

Response:

```
"success": true,
"data": {
    "id": "cmgki7fkg0001vsfg6ghz142f",
    "title": "Task title",
    "description": "Detailed task description",
    "priority": "MEDIUM",
    "status": "PLANNED",
    "category": "Category name",
    "tags": "[\"tag1\", \"tag2\"]",
    "createdAt": "2025-10-15T03:00:00.000Z",
    "updatedAt": "2025-10-15T03:00:00.000Z",
    "completedAt": null,
    "documents": []
}
```

Example API Calls with Different Priority Levels:

1. Create a LOW priority task:

```
curl -X POST http://localhost:3000/api/todos \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Update documentation styling",
    "description": "Improve markdown formatting in README files",
    "priority": "LOW",
    "status": "PLANNED",
    "category": "Enhancement",
    "tags": ["documentation", "style"]
}'
```

2. Create a MEDIUM priority task (default):

```
curl -X POST http://localhost:3000/api/todos \
   -H "Content-Type: application/json" \
   -d '{
     "title": "Add unit tests for TODO API",
     "description": "Create comprehensive test suite for TODO endpoints",
     "priority": "MEDIUM",
     "category": "Testing & QA",
     "tags": ["testing", "api"]
}'
```

3. Create a HIGH priority task:

```
curl -X POST http://localhost:3000/api/todos \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Optimize database queries",
    "description": "Profile and optimize slow database queries affecting performance",
    "priority": "HIGH",
    "status": "PLANNED",
    "category": "Performance",
    "tags": ["database", "optimization", "high-priority"]
}'
```

4. Create a CRITICAL priority task:

```
curl -X POST http://localhost:3000/api/todos \
  -H "Content-Type: application/json" \
  -d '{
    "title": "CRITICAL: Fix authentication bypass vulnerability",
    "description": "Security vulnerability discovered in authentication flow allowing unauthorized access",
    "priority": "CRITICAL",
    "status": "IN_PROGRESS",
    "category": "Security",
    "tags": ["security", "critical", "urgent", "blocking"]
}'
```

JavaScript/TypeScript Example:

```
// Using fetch API
async function createTodo(todoData: {
 title: string;
 description?: string;
 priority?: 'LOW' | 'MEDIUM' | 'HIGH' | 'CRITICAL';
 status?: 'PLANNED' | 'IN PROGRESS' | 'TESTING' | 'COMPLETE';
  category?: string;
 tags?: string[];
}) {
  const response = await fetch('/api/todos', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    body: JSON.stringify(todoData),
 });
  const result = await response.json();
  return result;
// Example usage
const newTodo = await createTodo({
 title: 'Implement feature X',
 description: 'Add new feature to the system',
 priority: 'HIGH',
category: 'Feature',
 tags: ['frontend', 'ui']
});
```

PUT /api/todos/[id] - Update TODO

Update an existing TODO item.

Request Body: Same as POST (all fields optional except those you want to update)

Example cURL:

```
curl -X PUT http://localhost:3000/api/todos/cmgki7fkg0001vsfg6ghz142f \
   -H "Content-Type: application/json" \
   -d '{
      "status": "IN_PROGRESS",
      "priority": "HIGH"
}'
```

DELETE /api/todos/[id] - Delete TODO

Remove a TODO item from the system.

Example cURL:

```
curl -X DELETE http://localhost:3000/api/todos/cmgki7fkg0001vsfg6ghz142f
```

POST /api/todos/[id]/complete - Mark TODO as complete

Mark a TODO as complete and set the completion timestamp.

Example cURL:

```
curl -X POST http://localhost:3000/api/todos/cmgki7fkg0001vsfg6ghz142f/complete
```

Authentication & Authorization

Current Status: No authentication required

The TODO API currently does not require authentication or authorization. All endpoints are publicly accessible on the local network. This is suitable for internal use within a trusted network environment.

Security Considerations:

- API is accessible to anyone on the same network
- Suitable for internal sports bar management systems
- For production internet-facing deployments, consider adding:
- JWT-based authentication
- Role-based access control (RBAC)
- API rate limiting
- IP whitelisting

GitHub Synchronization

The TODO system automatically synchronizes with GitHub:

- When a TODO is created, updated, or deleted, the TODO LIST.md file is automatically regenerated
- Changes are committed to the repository with descriptive commit messages
- Synchronization happens in the background without blocking API responses
- If GitHub sync fails, the operation still succeeds locally (sync errors are logged)

Sync Commit Messages:

- Create: chore: Add TODO - [Task Title]

```
- Update: chore: Update TODO - [Task Title]- Delete: chore: Remove TODO - [Task Title]
```

Best Practices

- 1. Always use the API Never edit TODO LIST.md directly
- 2. Use descriptive titles Make tasks easy to understand at a glance
- 3. Add detailed descriptions Include steps, affected components, and expected outcomes
- 4. Tag appropriately Use consistent tags for filtering and organization
- 5. Set correct priority Use CRITICAL sparingly for true blocking issues
- 6. Update status regularly Keep task status current as work progresses
- 7. **Complete tasks** Use the complete endpoint to properly timestamp completion

Backup & Maintenance

Automated Daily Backup

Schedule: Daily at 3:00 AM (server time)

Script: /home/ubuntu/github_repos/Sports-Bar-TV-Controller/backup_script.js
Backup Directory: /home/ubuntu/github_repos/Sports-Bar-TV-Controller/backups/

Retention: 14 days

Cron Job:

```
0 3 * * * cd /home/ubuntu/github_repos/Sports-Bar-TV-Controller && /usr/bin/node backup_script.js >> backup.log 2>&1
```

What Gets Backed Up:

- 1. Matrix configuration (JSON format)
- 2. Database files (prisma/data/sports_bar.db)
- 3. Atlas configurations

Backup File Format: backup_YYYY-MM-DD_HH-MM-SS.json

Manual Backup

Database:

```
pg_dump sports_bar_tv > backup_$(date +%Y%m%d_%H%M%S).sql
```

Application:

```
\label{tar-czf} tar -czf sports-bar-backup-\$(date + \$Y\$m\%d).tar.gz \sim /github\_repos/Sports-Bar-TV-Controller
```

Restore from Backup

Database:

```
psql sports_bar_tv < backup_20251015_020000.sql
```

Atlas Configuration:

```
cd ~/github_repos/Sports-Bar-TV-Controller/data/atlas-configs
cp cmgjxa5ai000260a7xuiepjl_backup_TIMESTAMP.json cmgjxa5ai000260a7xuiepjl.json
```

Troubleshooting

Application Issues

Application Won't Start:

```
# Check PM2 status
pm2 status

# View logs
pm2 logs sports-bar-tv

# Restart application
pm2 restart sports-bar-tv
```

Database Connection Errors:

```
# Check database status
npx prisma db pull

# Run pending migrations
npx prisma migrate deploy

# Regenerate Prisma client
npx prisma generate
```

Network Issues

Wolfpack Matrix Not Responding:

- Check network connectivity: ping <wolfpack-ip>
- 2. Verify matrix is powered on
- 3. Check network cable connections
- 4. Confirm same network/VLAN
- 5. Test connection in System Admin

Atlas Processor Offline:

- 1. Check connectivity: ping 192.168.5.101
- 2. Verify processor is powered on
- 3. Check configuration file exists
- 4. Restore from backup if needed

Performance Issues

Slow Response Times:

- 1. Check PM2 resource usage: pm2 monit
- 2. Review application logs

- 3. Check database size and optimize
- 4. Restart application if needed

High Memory Usage:

- 1. Check PM2 status: pm2 status
- 2. Restart application: pm2 restart sports-bar-tv
- 3. Monitor logs for memory leaks

Security Best Practices

Network Security

- · Wolfpack matrix on isolated VLAN
- · Application behind firewall
- Use HTTPS in production (configure reverse proxy)

Authentication

- Strong passwords for all accounts
- Regular password rotation
- Secure storage of credentials

API Security

- API keys in .env file only
- Never commit .env to repository
- Masked display in UI
- Server-side validation

Database Security

- Strong database passwords
- Restrict access to localhost
- · Regular security updates
- Encrypted backups

Recent Changes

October 15, 2025 - DirecTV Integration Documentation Update

Status: V Documentation complete

Updated By: DeepAgent

Documentation Added

- ✓ Comprehensive DirecTV Integration section (Section 7)
- Complete testing results from October 15, 2025 testing session
- Detailed error messages and diagnostics
- Verbose logging implementation details
- API endpoint specifications with request/response examples
- ✓ Database schema for DirecTVReceiver model

- <a> Known issues and limitations documentation
- Comprehensive troubleshooting guide
- V Production deployment recommendations
- V Network configuration guidelines
- Security considerations
- V Future enhancement roadmap

Testing Results Documented

Successful Operations:

- Receiver creation with full configuration
- Receiver deletion (tested with 9 receivers)
- Form validation and React integration
- Matrix input channel selection (32 channels)

Known Issues:

- Subscription polling requires physical DirecTV hardware
- Connection status ambiguity in UI
- Form input React state synchronization workaround needed
- Network topology dependencies

Logging Details Added

- PM2 log locations and access methods
- Logged operations for all DirecTV activities
- · Example log outputs for debugging
- Log search commands for troubleshooting

Troubleshooting Guide Includes

- Network connectivity verification steps
- Receiver status checking procedures
- Configuration validation methods
- Backend log review commands
- Firewall/port testing procedures
- Common problems and solutions

October 15, 2025 - PR #193: Unified Prisma Client & Al Hub Fixes (MERGED TO MAIN)

Status: V Successfully merged, tested, and deployed

Changes Implemented

1. Prisma Client Singleton Pattern

- ✓ Unified all Prisma client imports across 9 API route files to use singleton from @/lib/db
- Prevents multiple Prisma client instances and potential memory leaks
- Improves database connection management
- V Standardizes database access patterns throughout the application

2. Al Hub Database Models

- Added IndexedFile model for tracking indexed codebase files
- Added QAEntry model (renamed from QAPair) for Q&A training data

- Added TrainingDocument model for Al Hub training documents
- Added ApiKey model for managing Al provider API keys
- Successfully migrated database with new schema

3. Logging Enhancements

- Added comprehensive verbose logging to Al system components:
 - Codebase indexing process with file counts and progress
 - Vector embeddings generation and storage
 - Q&A entry creation with detailed field logging
 - Q&A entry retrieval with query debugging
 - Database operations with success/failure tracking

4. Bug Fixes

- V Fixed Q&A entries GET handler that was incorrectly processing requests as POST
- Corrected async/await patterns in Q&A API routes
- Improved error handling with detailed error messages

Testing Results (Remote Server: 24.123.87.42:3000)

All features successfully tested on production server:

- Codebase Indexing: 720 files successfully indexed
- **Q&A Entry Creation:** Successfully created test entries with proper field mapping
- V Q&A Entry Retrieval: GET requests now working correctly, returns all entries
- Verbose Logging: Confirmed in PM2 logs with detailed debugging information
- V Database Integrity: All migrations applied successfully, schema validated

Files Modified in PR #193

- src/app/api/ai-assistant/index-codebase/route.ts Added verbose logging & unified Prisma
- src/app/api/ai-assistant/search-code/route.ts Unified Prisma client import
- src/app/api/ai/qa-entries/route.ts Fixed GET handler bug & added logging
- src/app/api/cec/discovery/route.ts Unified Prisma client import
- src/app/api/home-teams/route.ts Unified Prisma client import
- src/app/api/schedules/[id]/route.ts Unified Prisma client import
- src/app/api/schedules/execute/route.ts Unified Prisma client import
- src/app/api/schedules/logs/route.ts Unified Prisma client import
- src/app/api/schedules/route.ts Unified Prisma client import
- prisma/schema.prisma Added new Al Hub models
- All backup files removed for clean codebase

Related Pull Requests

- **PR #193** Successfully merged to main branch (supersedes PR #169)
- X PR #169 Closed due to merge conflicts (superseded by PR #193)

Benefits Achieved

- Eliminated Prisma client instance conflicts
- Improved AI Hub reliability and debuggability
- Enhanced production monitoring with verbose logging
- Fixed critical Q&A entry retrieval bug
- Clean, maintainable codebase with consistent patterns

October 15, 2025 - Al Hub Testing & Documentation Update

- Comprehensive AI Hub testing completed
- V Identified 2 critical errors requiring immediate fixes
- Created detailed testing report
- Reorganized documentation by site tabs
- V Updated port from 3001 to 3000
- 🗸 Added detailed AI Hub section with status and fix plans

October 14, 2025 - Al Hub Database Models

- 🗸 Added missing database models (IndexedFile, QAPair, TrainingDocument, ApiKey)
- V Fixed Al Hub API routes
- Verified basic AI Hub functionality

October 10, 2025 - Atlas Configuration Restoration

- V Fixed critical Atlas configuration wipe bug
- Restored Atlas configuration from backup
- V Fixed dynamic zone labels
- V Implemented matrix label updates
- V Fixed matrix test database errors

October 9, 2025 - Outputs Configuration & Backup System

- Configured outputs 1-4 as full matrix outputs
- <a>Implemented automated daily backup system
- Added 14-day retention policy

Support Resources

Documentation Links

- Next.js: https://nextjs.org/docs
- Prisma: https://www.prisma.io/docs
- Tailwind CSS: https://tailwindcss.com/docs

Project Resources

- GitHub Repository: https://github.com/dfultonthebar/Sports-Bar-TV-Controller
- **GitHub Issues:** https://github.com/dfultonthebar/Sports-Bar-TV-Controller/issues
- Al Hub Testing Report: /home/ubuntu/ai hub testing report.md

Getting Help

- 1. Check this documentation
- 2. Review application logs: pm2 logs sports-bar-tv
- 3. Check GitHub issues
- 4. Create new issue with detailed information

Last Updated: October 15, 2025 by DeepAgent

Version: 2.2

Status: Production Ready (Al Hub has 2 critical issues requiring fixes)

Recent Deployments

PR #193 - Prisma Client Singleton Pattern Fix (October 15, 2025)

Deployment Date: October 15, 2025

Deployed By: DeepAgent

Commit: f51616d - "Fix: Unify Prisma Client Singleton Pattern (#193)"

Changes:

- Unified Prisma Client singleton pattern across the application

- Fixed database connection handling issues
- Improved application stability and performance

Deployment Steps Executed:

- 1. SSH connection established to production server (24.123.87.42:224)
- 2. Navigated to /home/ubuntu/Sports-Bar-TV-Controller
- 3. Pulled latest changes from main branch (already up to date)
- 4. Ran npm install (dependencies up to date)
- 5. Generated Prisma Client with npx prisma generate
- 6. Built application with npm run build (completed successfully)
- 7. Restarted PM2 process sports-bar-tv

Verification:

- PM2 process status: Online ✓
- Application startup: Successful (Ready in 496ms) ✓
- Memory usage: 57.0mb (healthy) ✓
- CPU usage: 0% (stable) ✓
- Uptime: Stable with no crashes ✓

Documentation Updates:

- Corrected production server path to /home/ubuntu/Sports-Bar-TV-Controller
- Updated PM2 process name to sports-bar-tv (was incorrectly documented as sports-bar-tv-controller)
- Added npx prisma generate step to deployment procedure
- Clarified distinction between development and production paths

11. Amazon Fire TV Integration

Overview

The Sports Bar TV Controller includes comprehensive integration with Amazon Fire TV devices for remote control, automation, and matrix routing. The system uses Android Debug Bridge (ADB) for network-based control of Fire TV devices, enabling automated content selection, app launching, and coordination with the Wolfpack HDMI matrix switcher.

Current Production Configuration:

- Fire TV Model: Fire TV Cube (3rd Gen - AFTGAZL)

- IP Address: 192.168.5.131- Port: 5555 (ADB default)- Matrix Input: Channel 13

- Connection Status: Fully operational - ADB Status: Enabled and connected

Fire TV Cube Specifications

Hardware:

- Model: AFTGAZL (Amazon Fire TV Cube, 3rd Generation - 2022)

- Processor: Octa-core ARM-based processor

- RAM: 2GB - Storage: 16GB

- Operating System: Fire OS 7+ (Based on Android 9)

- Network: Wi-Fi 6, Gigabit Ethernet port

- Ports: HDMI 2.1 output, Micro USB, Ethernet, IR extender

Capabilities:

- 4K Ultra HD, HDR, HDR10+, Dolby Vision
- Dolby Atmos audio
- Hands-free Alexa voice control
- Built-in speaker for Alexa
- IR blaster for TV control
- HDMI-CEC support

ADB Bridge Configuration

Server Configuration:

- ADB Path: /usr/bin/adb

- ADB Version: 1.0.41 (Version 28.0.2-debian)

Installation Location: /usr/lib/android-sdk/platform-tools/adb
 Connection Status: Active and connected to 192.168.5.131:5555

- **Device State:** "device" (fully operational)

- Setup Date: October 16, 2025

Connection Management:

```
# Connect to Fire TV Cube
adb connect 192.168.5.131:5555

# Check connection status
adb devices
# Expected output:
# List of devices attached
# 192.168.5.131:5555 device

# Test device communication
adb -s 192.168.5.131:5555 shell getprop ro.product.model
# Expected output: AFTGAZL

# Disconnect (if needed)
adb disconnect 192.168.5.131:5555
```

Enabling ADB on Fire TV

Step-by-Step Process:

1. Enable Developer Options:

- Go to Settings → My Fire $TV \rightarrow About$
- Click on device name 7 times rapidly
- "Developer Options" will appear in Settings

2. Enable ADB Debugging:

- Go to Settings → My Fire TV → Developer Options
- Turn on "ADB Debugging"
- Confirm warning dialog

3. First Connection Authorization:

- First ADB connection shows authorization prompt on TV
- Select "Always allow from this computer"
- Tap OK to authorize

Matrix Integration

Physical Connection:

- Fire TV Cube HDMI output → Wolfpack Matrix Input 13
- Matrix can route Input 13 to any of 32 TV outputs

Routing Control:

Benefits:

- Unified control of Fire TV content routing
- Show same Fire TV content on multiple displays
- Quick switching between Fire TV and other sources
- Coordinate Fire TV control with matrix routing

API Endpoints

Device Management

GET /api/firetv-devices

- Retrieve all configured Fire TV devices
- Returns array of device objects with status

POST /api/firetv-devices

- Add new Fire TV device

- Requires: name, ipAddress, port, deviceType
- Optional: inputChannel (matrix input)
- Validates IP format and prevents duplicates

PUT /api/firetv-devices

- Update existing Fire TV device configuration
- Can modify all fields except device ID

DELETE /api/firetv-devices?id={deviceId}

- Remove Fire TV device from system
- Device can be re-added anytime

Remote Control

POST /api/firetv-devices/send-command

Send remote control command to Fire TV:

```
{
  "deviceId": "device_id",
  "ipAddress": "192.168.5.131",
  "port": 5555,
  "command": "HOME"
}
```

Supported Commands:

- Navigation: UP, DOWN, LEFT, RIGHT, OK, BACK, HOME, MENU
- Media: PLAY_PAUSE, PLAY, PAUSE, REWIND, FAST_FORWARD
- Volume: VOL UP, VOL DOWN, MUTE
- Power: SLEEP, WAKEUP

App Launch:

```
{
  "deviceId": "device_id",
  "ipAddress": "192.168.5.131",
  "port": 5555,
  "appPackage": "com.espn.score_center"
}
```

Connection Testing

POST /api/firetv-devices/test-connection

Test connectivity to Fire TV device:

```
{
  "ipAddress": "192.168.5.131",
  "port": 5555,
  "deviceId": "device_id"
}
```

Success Response:

```
"success": true,
"message": "Fire TV device connected via ADB",
"deviceInfo": {
    "model": "AFTGAZL",
    "version": "Fire OS 7.6.6.8"
}
```

Subscription Polling

POST /api/firetv-devices/subscriptions/poll

Retrieve device status and installed apps:

```
{
  "deviceId": "device_id",
  "ipAddress": "192.168.5.131",
  "port": 5555
}
```

Remote Control Commands

Navigation Commands:

```
# D-Pad Navigation
                                                                     # 19
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE DPAD UP
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_DPAD_DOWN
                                                                     # 20
                                                                     # 21
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE DPAD LEFT
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_DPAD_RIGHT
                                                                     # 22
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE DPAD CENTER # 23 (OK/Select)
# System Navigation
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_HOME
                                                                     # 3
                                                                     # 4
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_BACK
                                                                     # 82
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_MENU
```

Media Controls:

```
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_MEDIA_PLAY_PAUSE # 85 adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_MEDIA_PLAY # 126 adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_MEDIA_PAUSE # 127 adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_MEDIA_REWIND # 89 adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_MEDIA_FAST_FORWARD # 90
```

Volume Controls:

```
adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_VOLUME_UP # 24 adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_VOLUME_DOWN # 25 adb -s 192.168.5.131:5555 shell input keyevent KEYCODE_VOLUME_MUTE # 164
```

Streaming Apps Configuration

Sports Streaming Apps:

```
- ESPN - com.espn.score center
```

```
- FOX Sports - com.fox.now
```

```
- NBC Sports - com.nbc.nbcsports.liveextra
- Paramount+ (CBS Sports) - com.cbs.ott
- YouTube TV - com.google.android.youtube.tv

League-Specific Apps:
- NFL+ - com.nflmobile.nflnow
- NFL Game Pass - com.nfl.gamepass
- NBA League Pass - com.nba.game
- MLB.TV - com.bamnetworks.mobile.android.gameday.mlb
- NHL.TV - com.nhl.gc1112.free
```

App Launch Commands:

```
# Launch ESPN
adb -s 192.168.5.131:5555 shell monkey -p com.espn.score_center 1

# Launch Netflix
adb -s 192.168.5.131:5555 shell monkey -p com.netflix.ninja 1

# Alternative method
adb -s 192.168.5.131:5555 shell am start -n com.espn.score_center/.MainActivity
```

Automation Capabilities

Scheduled App Launching:

```
# Crontab example: Launch ESPN at 7 PM daily
0 19 * * * curl -X POST http://localhost:3000/api/firetv-devices/send-command \
    -H "Content-Type: application/json" \
    -d '{"deviceId":"device_id","appPackage":"com.espn.score_center"}'
```

Coordinated Control:

```
# Script for game day setup
#!/bin/bash
# 1. Route Fire TV to main bar TVs
curl -X POST http://localhost:3000/api/matrix/route \
    -d '{"input":13,"outputs":[33,34,35]}'

# 2. Launch sports app
curl -X POST http://localhost:3000/api/firetv-devices/send-command \
    -d '{"deviceId":"device_id","appPackage":"com.espn.score_center"}'
```

Diagnostic Commands

Device Information:

```
# Get device model
adb -s 192.168.5.131:5555 shell getprop ro.product.model
# Output: AFTGAZL

# Get Fire OS version
adb -s 192.168.5.131:5555 shell getprop ro.build.version.release
# Output: 9

# Get all properties
adb -s 192.168.5.131:5555 shell getprop

# Get device uptime
adb -s 192.168.5.131:5555 shell uptime
```

Network Information:

```
# IP address details
adb -s 192.168.5.131:5555 shell ifconfig wlan0

# Network interfaces
adb -s 192.168.5.131:5555 shell ip addr show
```

Installed Apps:

```
# List all packages
adb -s 192.168.5.131:5555 shell pm list packages

# User-installed apps only
adb -s 192.168.5.131:5555 shell pm list packages -3

# Search for specific app
adb -s 192.168.5.131:5555 shell pm list packages | grep -i espn
```

Current App Status:

```
# Get currently focused app
adb -s 192.168.5.131:5555 shell dumpsys window | grep mCurrentFocus
# Output: mCurrentFocus=Window{hash u0 package/activity}
```

Troubleshooting

Device Shows Offline:

1. Verify Network Connectivity:

```
bash
ping 192.168.5.131
# Should respond with low latency (< 50ms)</pre>
```

2. Check ADB Status on Fire TV:

- Settings → My Fire TV → Developer Options
- Ensure "ADB Debugging" is ON
- May auto-disable after system updates

3. Test Port Accessibility:

bash

```
telnet 192.168.5.131 5555
# Or
nc -zv 192.168.5.131 5555
```

4. Restart ADB Server:

```
bash
```

```
adb kill-server
adb start-server
adb connect 192.168.5.131:5555
```

5. Restart Fire TV:

- Unplug Fire TV Cube for 30 seconds
- Plug back in, wait for full boot (2 minutes)
- Reconnect ADB

Commands Timeout:

- 1. Check network latency (should be < 100ms)
- 2. Verify Fire TV not overloaded (close background apps)
- 3. Test with simple command (HOME) first
- 4. Review PM2 logs for specific errors

Connection Drops:

1. Use Static IP:

- Assign static IP to Fire TV: 192.168.5.131
- Or use DHCP reservation based on MAC address

2. Improve Network:

- Use Ethernet instead of Wi-Fi (recommended)
- Check for network congestion
- Verify no VLAN isolation

3. Keep-Alive Script:

```
bash
```

```
# Run every 5 minutes via cron
*/5 * * * * adb -s 192.168.5.131:5555 shell echo "keepalive" > /dev/null
```

Health Monitoring

Automated Health Check Script:

```
#!/bin/bash
# /home/ubuntu/scripts/firetv-health-check.sh

DEVICE_IP="192.168.5.131"
DEVICE_PORT="5555"
LOG="/var/log/firetv-health.log"

# Test connectivity
if ! adb devices | grep "$DEVICE_IP:$DEVICE_PORT" | grep "device" > /dev/null; then
    echo "[$(date)] Fire TV offline - reconnecting" >> $LOG
    adb connect $DEVICE_IP:$DEVICE_PORT >> $LOG
else
    echo "[$(date)] Fire TV online" >> $LOG
fi
```

Schedule with cron:

```
# Run every 5 minutes
*/5 * * * * /home/ubuntu/scripts/firetv-health-check.sh
```

Monitoring Metrics:

- Connection status (online/offline)
- Response time (should be < 500ms)
- Command success rate (target > 95%)
- ADB connection stability

Data Storage

Configuration File:

- **Location:** /data/firetv-devices.json
- Format: JSON array of device objects
- Backup: Included in automated daily backups (3:00 AM)
- Backup Location: /backups/ with timestamps

Device Object Structure:

```
"id": "firetv_timestamp_hash",
    "name": "Fire TV Cube Bar",
    "ipAddress": "192.168.5.131",
    "port": 5555,
    "deviceType": "Fire TV Cube",
    "inputChannel": "13",
    "isOnline": true,
    "adbEnabled": true,
    "addedAt": "2025-10-16T10:30:00.000Z",
    "updatedAt": "2025-10-16T12:00:00.000Z"
}
```

Best Practices

Network Configuration:

- Use static IP address for Fire TV devices
- Ethernet connection preferred over Wi-Fi

- Ensure low latency (< 50ms) to Fire TV
- No firewall blocking port 5555

Device Management:

- Keep ADB debugging enabled at all times
- Regular connectivity tests before events
- Monitor for system updates (may disable ADB)
- Restart Fire TV weekly during maintenance

Security:

- Limit access to ADB port (5555) from external networks
- Use network segmentation for streaming devices
- Secure SSH access to controller server
- Monitor unauthorized ADB connection attempts

Performance:

- Close unused apps regularly
- Clear cache monthly
- Monitor storage space (keep > 2GB free)
- Restart Fire TV during off-hours

Integration with Sports Guide

Automated Content Selection:

- Sports Guide can trigger Fire TV app launches
- Route Fire TV to appropriate displays based on schedule
- Coordinate multiple Fire TVs for multi-game viewing
- Automatic switching between streaming services

Game Day Workflow:

- 1. Sports Guide identifies upcoming games
- 2. System determines which streaming service has content
- 3. Fire TV launches appropriate app
- 4. Matrix routes Fire TV to designated displays
- 5. Content ready for viewing at game time

Documentation References

Comprehensive Documentation:

- Q&A Sheet: /home/ubuntu/amazon_firetv_qa_sheet.md
- **ADB Bridge Setup:** /home/ubuntu/firetv_ads_bridge_setup.md
- **Testing Report:** /home/ubuntu/firetv_testing_findings.md
- Total Q&A Pairs: 95+ covering all aspects

Topics Covered:

- Device setup and configuration
- ADB bridge installation and setup
- Matrix integration
- API endpoints reference
- Remote control commands
- Troubleshooting procedures
- Best practices for deployment

Current Status

Production Environment:

- V Fire TV Cube connected and operational (192.168.5.131:5555)
- ADB bridge fully configured and tested
- Matrix integration active (Input 13)
- API endpoints operational
- Remote control commands working
- Form submission bugs fixed (October 15, 2025)
- CSS styling issues resolved
- Comprehensive documentation complete

Last Updated: October 16, 2025 **Last Tested:** October 16, 2025 **Status:** Production Ready ✓

LATEST UPDATE: Sports Guide v5.0.0 - October 16, 2025

Critical Fix Applied

Issue: Sports Guide was not loading ANY data from The Rail Media API.

Root Cause: Frontend/backend parameter mismatch - frontend sent selectedLeagues , backend expected days/startDate/endDate .

Solution: Drastically simplified the entire system:

- REMOVED all league selection UI (800+ lines of code removed)
- ADDED automatic loading of ALL sports on page visit
- **ADDED** maximum verbosity logging for Al analysis
- V FIXED both Sports Guide and Bartender Remote Channel Guide

Results

Sports Guide (/sports-guide):

- Auto-loads 7 days of ALL sports programming
- Displays 17+ sports categories
- Shows 361+ games with full details
- No user interaction required
- Simple search and refresh interface

Bartender Remote Channel Guide (/remote → **Guide tab)**:

- V Successfully loads sports data from Rail Media API
- Shows device-specific channel numbers
- ✓ Cable/Satellite/Streaming support
- Watch" button integration

Testing Verified

Test Date: October 16, 2025 at 4:29-4:30 AM

Sports Guide Test Results:

- Loaded 17 sports, 361 games
- MLB Baseball: 18 games
- NBA Basketball: 22 games
- NFL, NHL, College sports, Soccer, and more
- Load time: ~5 seconds

Bartender Remote Test Results:

- Cable Box 1 guide loaded successfully
- MLB games displayed with channel numbers (FOXD 831, UniMas 806)
- NBA games displayed with channel numbers (ESPN2 28, NBALP)
- Watch buttons functional

Architecture Changes (v5.0.0)

Before:

```
Frontend → { selectedLeagues: [...] }

API Route expects { days, startDate, endDate }

MISMATCH X → No data
```

After:

```
Frontend Auto-load on mount (days: 7)

API Route Fetch from Rail Media API

ALL sports data returned 

Display in both Sports Guide and Bartender Remote
```

Maximum Verbosity Logging

All API routes now include comprehensive timestamped logging:

- Every API request with full parameters
- Full API responses with data counts
- Error details with stack traces
- Performance timing
- Accessible via pm2 logs sports-bar-tv

Files Modified

- 1. /src/app/api/sports-guide/route.ts Simplified auto-loading API
- 2. /src/components/SportsGuide.tsx Removed league UI, added auto-loading
- 3. /src/app/api/channel-guide/route.ts Integrated Rail Media API

Deployment

```
cd /home/ubuntu/Sports-Bar-TV-Controller
npm run build
pm2 restart sports-bar-tv
```

Status: Application successfully rebuilt and restarted.

Detailed Report

See SPORTS_GUIDE_FIX_REPORT.md for complete technical details, testing results, and architecture diagrams.