

Sports Bar TV Controller - System Documentation

Version: 2.2

Last Updated: October 15, 2025

Status: Production Ready

Quick Access Information

Server Access

- **Host:** 24.123.87.42
- **Port:** 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
```

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.prisma`
- **Migrations:** `prisma/migrations/`

Key Database Models

- `MatrixOutput` - TV display outputs
 - `MatrixInput` - Video sources
 - `WolfpackConfig` - Matrix switcher configuration
 - `AudioProcessor` - Atlas audio configuration
 - `IndexedFile` - AI Hub codebase files
 - `QAPair` - AI Hub Q&A training data
 - `TrainingDocument` - AI Hub training documents
 - `ApiKey` - AI provider API keys
 - `TODD` - 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:** PostgreSQL
 - **Hardware Integration:**
 - Wolfpack HDMI Matrix Switchers (via HTTP API)
 - Atlas AZMP8 Audio Processor (via HTTP API)
 - **Process Management:** PM2
 - **AI Integration:** Multiple AI providers (Ollama, Abacus AI, OpenAI, Anthropic, X.AI)
-

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:**
 - AI Hub - Unified AI 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

```
model MatrixOutput {
  id          Int          @id @default(autoincrement())
  outputNumber Int          @unique
  label       String
  enabled     Boolean      @default(true)
  isActive    Boolean      @default(false)
  currentInput Int?
  audioOutput  Int?
  resolution   String?
  dailyTurnOn  Boolean      @default(true)
  dailyTurnOff Boolean      @default(true)
  isMatrixOutput Boolean    @default(true)
  createdAt    DateTime     @default(now())
  updatedAt    DateTime     @updatedAt
}
```

MatrixInput

```
model MatrixInput {
  id          Int          @id @default(autoincrement())
  inputNumber Int          @unique
  label       String
  enabled     Boolean      @default(true)
  createdAt    DateTime     @default(now())
  updatedAt    DateTime     @updatedAt
}
```

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 AI-powered analysis of audio system performance

Configuration Management

Configuration File Location:

- Primary: /home/ubuntu/github_repos/Sports-Bar-TV-Controller/data/atlas-configs/cmgyx-a5ai000260a7xuietj1.json
- Backups: /home/ubuntu/github_repos/Sports-Bar-TV-Controller/data/atlas-configs/cmgyx-a5ai000260a7xuietj1_backup_*.json

Backup Strategy:

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

Database Schema

```
model AudioProcessor {
  id          String @id @default(cuid())
  name        String
  model       String
  ipAddress   String
  port        Int @default(80)
  username    String?
  password    String?
  isActive    Boolean @default(true)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}
```

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. AI 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: ⚠️ **PARTIALLY FUNCTIONAL**

Critical Issues: 2

Features Tested: 7

Working Features: 5

Broken Features: 2

Features & Status

✅ AI Assistant Tab (Partially Working)

Status: Chat interface works, Codebase sync fails

Chat Interface:

- ✅ **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:

- ❌ **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:

- ✅ **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:

- ❌ **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:

- ✅ **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
- Q&A Pairs: 0
- Total Content: 0 Bytes
- Last Updated: 10/15/2025, 1:00:06 AM

✅ Enhanced Devices Tab (Working)

Status: ✅ 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 AI insights available for the selected criteria"

✓ Configuration Tab (Working)

Status: ✓ FULLY FUNCTIONAL

Provider Statistics:

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

Local AI Services:

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

- AI System Diagnostics (expandable)
- Provider status monitoring
- Refresh status button
- Local AI setup guide

✓ API Keys Tab (Working)

Status: ✓ FULLY FUNCTIONAL

Features:

- API key management interface
- Configured API Keys display (currently 0)
- Add API Key button
- Provider documentation links:
- Ollama (Local) - RECOMMENDED
- Abacus AI
- OpenAI
- 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          String    @id @default(cuid())
  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 @default(cuid())
  question    String    @db.Text
  answer      String    @db.Text
  context     String?   @db.Text
  source      String?
  category    String?
  isActive    Boolean   @default(true)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}

model TrainingDocument {
  id          String    @id @default(cuid())
  title       String
  content     String    @db.Text
  fileType    String
  fileSize    Int
  category    String?
  isActive    Boolean   @default(true)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}

model ApiKey {
  id          String    @id @default(cuid())
  provider    String
  keyName     String
  apiKey      String
  isActive    Boolean   @default(true)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  @@unique([provider, keyName])
}

```

API Endpoints

POST `/api/ai-assistant/index-codebase`

 **Status:** BROKEN (404 error)

Index codebase files for AI analysis

POST `/api/ai-assistant/chat`

 **Status:** WORKING (slow)

Chat with AI about codebase

POST `/api/ai/qa-generate`

Generate Q&A pairs from repository

POST `/api/ai/qa-entries`

 **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

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

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 AI 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

Integration with The Rail Media API for sports programming information.

Features

- Sports channel guide
- Programming schedules
- Event information
- API key management with validation

API Configuration

Provider: The Rail Media

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

Current User ID: 258351

API Endpoints

GET `/api/sports-guide/status`

Get current API configuration status

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

Verify API key validity

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

Update API key (with validation)

GET `/api/sports-guide/channels`

Fetch channel guide data with filtering options

Configuration

1. Navigate to Sports Guide Configuration
2. Click "API" tab
3. Enter User ID and API Key
4. Click "Verify API Key" to test
5. System validates before saving
6. Restart server for full effect

Security

- API keys stored in `.env` file (not in repository)
 - Keys masked in UI (shows first 8 and last 4 characters only)
 - Validation before saving
 - Server-side API calls only
-

6. Streaming Platforms

Overview

Management interface for streaming service accounts and configurations.

Features

- Platform account management
 - Service configuration
 - Integration settings
-

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)
- ☒ Program guide data for specific channels
- ☒ Device location information (multi-room setups)

Protocol Limitations:

- ☒ NO subscription/package information
- ☒ NO account details or billing data
- ☒ NO entitled channels list
- ☒ 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

- ✓ Device connectivity testing
- ✓ Real-time device status monitoring
- ✓ Current channel and program information
- ✓ 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", "chardown")
- 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, chardown, 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

✗ Failed Operations & Known Issues

1. Subscription Data Retrieval (FAILED)

- **Status:** ✗ 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: 0
 - Sports 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-tv`
- **Log 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]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):

```
{
  "success": true,
  "activeSubscriptions": 150,
  "sportsPackages": 12,
  "packages": [
    {"name": "NFL Sunday Ticket", "active": true},
    {"name": "NBA League Pass", "active": true}
  ],
  "lastUpdated": "2025-10-15T18:10:26.000Z"
}
```

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 String @id @default(cuid())
  deviceName String
  ipAddress String
  port Int @default(8080)
  receiverType String
  matrixInputChannel Int
  connected Boolean @default(false)
  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. 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

9. 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 {
  id          String      @id @default(cuid())
  title       String
  description  String?
  priority    String      @default("MEDIUM") // "LOW", "MEDIUM", "HIGH", "CRITICAL"
  status      String      @default("PLANNED") // "PLANNED", "IN_PROGRESS", "TESTING", "COMPLETE"
  category    String?
  tags        String?     // JSON array of tags
  createdAt   DateTime     @default(now())
  updatedAt   DateTime     @updatedAt
  completedAt DateTime?

  documents   TodoDocument[]
}

model TodoDocument {
  id          String      @id @default(cuid())
  todoId      String
  filename    String
  filepath    String
  filesize    Int?
  mimetype    String?
  uploadedAt  DateTime     @default(now())

  todo        Todo        @relation(fields: [todoId], references: [id], onDelete: Cascade)

  @@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/cmgi7fkg0001vsfg6ghz142f \
-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/cmgi7fkg0001vsfg6ghz142f
```

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/cmgi7fkg0001vsfg6ghz142f/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:

```
tar -czf sports-bar-backup-$(date +%Y%m%d).tar.gz ~/github_repos/Sports-Bar-TV-Con-
troller
```

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

Wolfgang Matrix Not Responding:

1. Check network connectivity: `ping <wolfgang-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:  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

-  Known issues and limitations documentation
-  Comprehensive troubleshooting guide
-  Production deployment recommendations
-  Network configuration guidelines
-  Security considerations
-  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 & AI Hub Fixes (MERGED TO MAIN)



Status:  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
-  Standardizes database access patterns throughout the application

2. AI 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 AI Hub training documents
-  Added `ApiKey` model for managing AI provider API keys
-  Successfully migrated database with new schema

3. Logging Enhancements






-  Added comprehensive verbose logging to AI 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

-  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
-  **Q&A Entry Retrieval:** GET requests now working correctly, returns all entries
-  **Verbose Logging:** Confirmed in PM2 logs with detailed debugging information
-  **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 AI Hub models
- All backup files removed for clean codebase







Related Pull Requests

-  **PR #193** - Successfully merged to main branch (supersedes PR #169)
-  **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 - AI Hub Testing & Documentation Update

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




October 14, 2025 - AI Hub Database Models

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

October 10, 2025 - Atlas Configuration Restoration

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

October 9, 2025 - Outputs Configuration & Backup System

-  Configured outputs 1-4 as full matrix outputs
-  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>
- **AI 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 (AI 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