AI Diagnostics System Documentation

Overview

The AI Diagnostics System provides comprehensive monitoring, self-healing, and learning capabilities for the Sports Bar TV Controller application. It consists of three main components:

1. **Light Health Checks** - Quick 5-minute checks
2. **Deep Diagnostics** - Comprehensive Sunday 5 AM analysis
3. **Self-Healing** - Automatic issue resolution

Architecture

Database Schema

The system uses the following Prisma models to track system health:

SystemHealthCheck

Stores individual health check results:
- `checkType` : “light” or “deep”
- `component` : Which component was checked (pm2, api, database, etc.)
- `status` : “healthy”, “warning”, “critical”, or “error”
- `metrics` : JSON object with component-specific metrics
- `responseTime` : Time taken for the check

Issue

Tracks detected problems:
- `type` : crash, performance, resource, connectivity, dependency, security
- `severity` : low, medium, high, critical
- `status` : open, fixing, fixed, ignored
- `autoFixed` : Whether it was automatically resolved
- `fixAttempts` : Number of fix attempts

Fix

Records applied fixes:
- `action` : Type of fix applied
- `success` : Whether the fix worked
- `details` : JSON with execution details
- `duration` : Time taken to apply fix

SystemMetric

Historical metrics for trend analysis:
- `metricType` : cpu, memory, disk, response_time, etc.
- `value` : Metric value
- `unit` : Unit of measurement

LearningPattern

Identified patterns and predictions:
- `patternType`: recurring_issue, performance_trend, etc.
- `occurrences`: How many times seen
- `frequency`: hourly, daily, weekly, sporadic
- `recommendation`: Suggested preventive measure

DiagnosticRun

Summary of each diagnostic execution:
- `runType`: light, deep, manual
- `status`: completed, failed, partial
- `checksRun/Passed/Failed/Warning`: Statistics
- `issuesFound/Fixed`: Issue counts
- `recommendations`: JSON array of suggestions

Components

1. Light Health Check (`light-check.js`)

Runs every 5 minutes to monitor critical system health.

Checks Performed:
- PM2 process status (online, memory, CPU, restarts)
- API health endpoint availability
- Database connectivity and size
- Disk space usage (warns at 80%, critical at 90%)
- Memory usage (warns at 85%, critical at 95%)
- System load average

Usage:

```
node scripts/diagnostics/light-check.js
```

Exit Codes:
- 0: All checks passed
- 1: One or more checks failed

2. Deep Diagnostics (`deep-diagnostics.js`)

Runs every Sunday at 5:00 AM for comprehensive analysis.

Checks Performed:
- Full dependency audit (npm list)
- Security vulnerability scan (npm audit)
- Performance analysis (7-day metrics)
- Log file analysis (errors, warnings, size)
- Database integrity check (PRAGMA integrity_check)
- External integration testing (Wolf Pack, Atlas, CEC, etc.)
- Configuration validation
- Optimization recommendations

Usage:

```
node scripts/diagnostics/deep-diagnostics.js
```

Output:
Generates a comprehensive report with:
- Dependency status
- Security vulnerabilities
- Performance metrics
- Database health
- Integration status
- Optimization recommendations

3. Self-Healing (`self-healing.js`)

Automatically fixes detected issues.

Capabilities:

Crash Recovery

- Restart crashed PM2 processes
- Verify process stability after restart

Resource Management

- Clean disk space when usage > 90%
- Clear npm cache
- Delete old logs (>30 days)
- Remove temp files (>7 days)
- Handle high memory by restarting services

Dependency Fixes

- Reinstall missing npm packages
- Repair corrupted database (VACUUM)

Proactive Maintenance

- Rotate large log files (>50MB)
- Clean old diagnostic data (>30 days)

Usage:

```
node scripts/diagnostics/self-healing.js
```

Automatic Triggering:
Self-healing is automatically triggered when light checks detect issues.

4. Scheduler (`scheduler.js`)

Manages scheduled execution of diagnostics.

Schedule:
- Light checks: Every 5 minutes
- Deep diagnostics: Sunday 5:00 AM EST

Usage:

```
node scripts/diagnostics/scheduler.js
```

Running as PM2 Process:

```
pm2 start scripts/diagnostics/scheduler.js --name diagnostics-scheduler
pm2 save
```

Installation

1. Update Database Schema

Add the diagnostic models to your Prisma schema:

```
# Copy the models from prisma/schema-diagnostics.prisma to prisma/schema.prisma
# Then run:
npx prisma generate
npx prisma migrate dev --name add_diagnostics_models
```

2. Install Dependencies

```
npm install axios node-cron
```

3. Start the Scheduler

```
# Option 1: Run directly
node scripts/diagnostics/scheduler.js

# Option 2: Run with PM2 (recommended)
pm2 start scripts/diagnostics/scheduler.js --name diagnostics-scheduler
pm2 save
```

4. Verify Installation

```
# Run a manual light check
node scripts/diagnostics/light-check.js

# Check PM2 status
pm2 status
```

API Endpoints

Manual Diagnostics Trigger

Add these endpoints to your Next.js API:

```
// pages/api/diagnostics/light-check.ts
import { exec } from 'child_process';
import { promisify } from 'util';

const execAsync = promisify(exec);

export default async function handler(req, res) {
  if (req.method !== 'POST') {
    return res.status(405).json({ error: 'Method not allowed' });
  }

  try {
    const { stdout } = await execAsync(
      'node /home/ubuntu/Sports-Bar-TV-Controller/scripts/diagnostics/light-check.js'
    );

    res.status(200).json({
      success: true,
      output: stdout
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      error: error.message
    });
  }
}
```

```
// pages/api/diagnostics/deep.ts
// Similar implementation for deep diagnostics
```

```
// pages/api/diagnostics/self-heal.ts
// Similar implementation for self-healing
```

Monitoring Dashboard

View Recent Checks

```
// Get recent health checks
const recentChecks = await prisma.systemHealthCheck.findMany({
  take: 50,
  orderBy: { timestamp: 'desc' },
  include: { issue: true }
});
```

View Open Issues

```
// Get open issues
const openIssues = await prisma.issue.findMany({
  where: { status: 'open' },
  orderBy: [
    { severity: 'desc' },
    { timestamp: 'desc' }
  ]
});
```

View System Metrics

```
// Get metrics for last 24 hours
const dayAgo = new Date(Date.now() - 24 * 60 * 60 * 1000);

const metrics = await prisma.systemMetric.findMany({
  where: {
    timestamp: { gte: dayAgo }
  },
  orderBy: { timestamp: 'asc' }
});
```

View Learning Patterns

```
// Get active patterns
const patterns = await prisma.learningPattern.findMany({
  where: { isActive: true },
  orderBy: { lastSeen: 'desc' }
});
```

Configuration

Environment Variables

```
# .env
DATABASE_URL="file:./prisma/data/sports_bar.db"
API_BASE_URL="http://localhost:3000"
NODE_ENV="production"
```

Thresholds

Edit the CONFIG object in each script to adjust thresholds:

```
const CONFIG = {
  DISK_WARNING_THRESHOLD: 80,      // percent
  DISK_CRITICAL_THRESHOLD: 90,     // percent
  MEMORY_WARNING_THRESHOLD: 85,    // percent
  MEMORY_CRITICAL_THRESHOLD: 95,   // percent
  API_TIMEOUT: 5000,               // milliseconds
  MAX_LOG_SIZE: 100 * 1024 * 1024, // bytes
  DAYS_TO_ANALYZE: 7               // days
};
```

Troubleshooting

Scheduler Not Running

```
# Check PM2 status
pm2 status

# View logs
pm2 logs diagnostics-scheduler

# Restart
pm2 restart diagnostics-scheduler
```

Database Connection Issues

```
# Check database file exists
ls -lh /home/ubuntu/Sports-Bar-TV-Controller/prisma/data/sports_bar.db

# Test connection
node -e "const { PrismaClient } = require('@prisma/client'); const prisma = new Pris-
maClient(); prisma.\$queryRaw\`SELECT 1\`.then(() => con-
sole.log('OK')).catch(console.error);"
```

Permission Issues

```
# Ensure scripts are executable
chmod +x scripts/diagnostics/*.js

# Check file ownership
ls -la scripts/diagnostics/
```

Best Practices

1. **Monitor the Monitors**: Check scheduler logs regularly
   `bash`
   `   pm2 logs diagnostics-scheduler --lines 100`

2. **Review Weekly Reports**: Check deep diagnostic reports every Monday

3. **Act on Recommendations**: Implement optimization suggestions from deep diagnostics

4. **Backup Before Fixes**: Self-healing creates backups, but manual backups are recommended

5. **Test in Development**: Test diagnostic scripts in dev environment first

6. **Adjust Thresholds**: Fine-tune thresholds based on your system's normal behavior

Future Enhancements

- [ ] Email/Slack notifications for critical issues
- [ ] Machine learning for pattern prediction
- [ ] Automated performance optimization
- [ ] Integration with external monitoring tools

- [ ] Custom alert rules and webhooks
- [ ] Historical trend visualization
- [ ] Predictive maintenance scheduling

Support

For issues or questions:
1. Check logs: `pm2 logs diagnostics-scheduler`
2. Review database: Check `DiagnosticRun` table for recent runs
3. Manual test: Run scripts individually to isolate issues
4. Check permissions: Ensure proper file and directory permissions

License

Part of the Sports Bar TV Controller project.