Media Center Developer Guide

Quick reference for implementing the new media center features using the updated Prisma schema.

Table of Contents

Importing Prisma Client

```
import { prisma } from '@/lib/db';
// or
import { PrismaClient } from '@prisma/client';
const prisma = new PrismaClient();
```

Streaming Platforms

Get All Active Streaming Platforms

```
// Get all platforms
const platforms = await prisma.streamingPlatform.findMany({
  where: { isActive: true },
  orderBy: { name: 'asc' }
});

// Get platforms by tier access
const freePlatforms = await prisma.streamingPlatform.findMany({
  where: {
    isActive: true,
    tierAccess: 'FREE'
  }
});

// Get premium platforms
const premiumPlatforms = await prisma.streamingPlatform.findMany({
  where: {
    isActive: true,
    tierAccess: { in: ['PREMIUM', 'PRO'] }
  }
});
```

Get Platform with User Favorites Count

```
const platformWithStats = await prisma.streamingPlatform.findUnique({
  where: { platformId: 'pickleballtv' },
  include: {
    userFavorites: {
      select: { userId: true }
    },
    _count: {
      select: { userFavorites: true }
    }
  }
});
```

Add/Remove User Favorite

```
// Add to favorites
await prisma.userFavoriteStreamingPlatform.create({
  data: {
    userId: session.user.id,
    platformId: 'pickleballtv'
  }
});

// Remove from favorites
await prisma.userFavoriteStreamingPlatform.delete({
  where: {
    userId_platformId: {
      userId: session.user.id,
      platformId: 'pickleballtv'
    }
  }
});

// Check if user favorited
const isFavorite = await prisma.userFavoriteStreamingPlatform.findUnique({
  where: {
    userId_platformId: {
      userId: session.user.id,
      platformId: 'pickleballtv'
    }
  }
});
```

Track Platform Views/Clicks

```
// Increment view count
await prisma.streamingPlatform.update({
  where: { id: platformId },
  data: { viewCount: { increment: 1 } }
});

// Increment click count
await prisma.streamingPlatform.update({
  where: { id: platformId },
  data: { clickCount: { increment: 1 } }
});
```

Tournament Organizations

Get All Organizations

```
const organizations = await prisma.tournamentOrganization.findMany({
  where: { isActive: true },
  include: {
    _count: {
      select: { userFollows: true }
    }
  },
  orderBy: { name: 'asc' }
});
```

Follow/Unfollow Organization

```
// Follow organization
await prisma.userFollowedOrganization.create({
  data: {
    userId: session.user.id,
    organizationId: orgId
  }
});

// Unfollow organization
await prisma.userFollowedOrganization.delete({
  where: {
    userId_organizationId: {
      userId: session.user.id,
      organizationId: orgId
    }
  }
});

// Get user's followed organizations
const followed = await prisma.userFollowedOrganization.findMany({
  where: { userId: session.user.id },
  include: { organization: true }
});
```

User Preferences

Get/Create User Media Preferences

```
// Get or create preferences
const preferences = await prisma.userMediaPreferences.upsert({
  where: { userId: session.user.id },
  create: {
    userId: session.user.id,
    eventReminderEnabled: true,
    tournamentReminderEnabled: true,
    podcastNewEpisodeNotification: true
  },
  update: {}
});
```

Update Preferences

```
await prisma.userMediaPreferences.update({
  where: { userId: session.user.id },
  data: {
    eventReminderEnabled: true,
    eventReminderTiming: 24, // hours before event
    weeklyMediaDigestEnabled: true,
    autoplayEnabled: false,
    preferredVideoQuality: 'hd'
  }
});
```

Manage Favorite Podcasts

```
// Add podcast to favorites
const currentPrefs = await prisma.userMediaPreferences.findUnique({
  where: { userId: session.user.id }
});

const favoritePodcasts = (currentPrefs?.favoritePodcasts as string[]) || [];
favoritePodcasts.push(podcastId);

await prisma.userMediaPreferences.update({
  where: { userId: session.user.id },
  data: {
    favoritePodcasts: favoritePodcasts
  }
});

// Remove podcast from favorites
const updatedFavorites = favoritePodcasts.filter(id => id !== podcastId);
await prisma.userMediaPreferences.update({
  where: { userId: session.user.id },
  data: {
    favoritePodcasts: updatedFavorites
  }
});
```

Trial Management

Check Trial Status

```
const user = await prisma.user.findUnique({
  where: { id: session.user.id },
  select: {
    subscriptionTier: true,
    isTrialActive: true,
    trialStartDate: true,
    trialEndDate: true
  }
});

if (user?.isTrialActive && user?.trialEndDate) {
  const daysLeft = Math.ceil(
    (user.trialEndDate.getTime() - new Date().getTime()) / (1000 * 60 * 60 * 24)
  );

  if (daysLeft <= 0) {
    // Trial expired
  } else if (daysLeft <= 3) {
    // Show urgent upgrade prompt
  }
}
```

Create Trial Expiration Reminders

```
// When user starts trial
const trialEndDate = new Date();
trialEndDate.setDate(trialEndDate.getDate() + 7); // 7-day trial

const reminderSchedules = [
  { type: 'TRIAL_7_DAYS_LEFT', days: 7 },
  { type: 'TRIAL_3_DAYS_LEFT', days: 3 },
  { type: 'TRIAL_1_DAY_LEFT', days: 1 },
  { type: 'TRIAL_EXPIRED', days: 0 }
];

for (const schedule of reminderSchedules) {
  const scheduledDate = new Date(trialEndDate);
  scheduledDate.setDate(scheduledDate.getDate() - schedule.days);

  await prisma.trialExpirationReminder.create({
    data: {
      userId: user.id,
      reminderType: schedule.type as any,
      reminderTiming: schedule.days,
      scheduledFor: scheduledDate,
      status: 'PENDING'
    }
  });
}
```

Get Pending Reminders

```
// Get reminders to send
const pendingReminders = await prisma.trialExpirationReminder.findMany({
  where: {
    status: 'PENDING',
    scheduledFor: { lte: new Date() },
    emailSent: false
  },
  include: {
    user: {
      select: {
        email: true,
        name: true,
        trialEndDate: true
      }
    }
  }
});
```

Mark Reminder as Sent

```
await prisma.trialExpirationReminder.update({
  where: { id: reminderId },
  data: {
    status: 'SENT',
    emailSent: true,
    sentAt: new Date(),
    emailStatus: 'SENT'
  }
});
```

Engagement Tracking

Track Any Media Interaction

```
async function trackEngagement({
  userId,
  sessionId,
  contentType,
  contentId,
  contentTitle,
  engagementType,
  metadata
}: {
  userId?: string;
  sessionId?: string;
  contentType: string;
  contentId: string;
  contentTitle?: string;
  engagementType: string;
  metadata?: any;
}) {
  await prisma.mediaEngagementMetrics.create({
    data: {
      userId,
      sessionId,
      contentType: contentType as any,
      contentId,
      contentTitle,
      engagementType: engagementType as any,
      metadata: metadata || {},
      timestamp: new Date()
    }
  });
}
```

Example Usage

```
// Track platform view
await trackEngagement({
  userId: session?.user?.id,
  sessionId: sessionId,
  contentType: 'STREAMING_PLATFORM',
  contentId: 'pickleballtv',
  contentTitle: 'PickleballTV',
  engagementType: 'VIEW',
  metadata: { source: 'media_center_page' }
});

// Track external link click
await trackEngagement({
  userId: session?.user?.id,
  contentType: 'STREAMING_PLATFORM',
  contentId: 'pickleballtv',
  engagementType: 'VISIT_WEBSITE',
  metadata: {
    url: 'https://pickleballtv.com',
    referrer: '/media/streaming'
  }
});

// Track organization follow
await trackEngagement({
  userId: session.user.id,
  contentType: 'TOURNAMENT_ORGANIZATION',
  contentId: orgId,
  contentTitle: 'PPA Tour',
  engagementType: 'FOLLOW'
});
```

Get Engagement Analytics

```
// Get top platforms by views
const topPlatforms = await prisma.mediaEngagementMetrics.groupBy({
  by: ['contentId', 'contentTitle'],
  where: {
    contentType: 'STREAMING_PLATFORM',
    engagementType: 'VIEW',
    timestamp: { gte: startDate }
  },
  _count: { id: true },
  orderBy: { _count: { id: 'desc' } },
  take: 10
});

// Get user engagement history
const userEngagement = await prisma.mediaEngagementMetrics.findMany({
  where: { userId: session.user.id },
  orderBy: { timestamp: 'desc' },
  take: 50
});

// Get platform-specific metrics
const platformMetrics = await prisma.mediaEngagementMetrics.findMany({
  where: {
    contentType: 'STREAMING_PLATFORM',
    contentId: platformId
  },
  select: {
    engagementType: true,
    timestamp: true,
    userId: true
  }
});
```

Example API Routes

`/api/media/streaming-platforms/route.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth';
import { prisma } from '@/lib/db';

export async function GET(req: NextRequest) {
  const session = await getServerSession(authOptions);

  // Get user's subscription tier
  const user = session?.user?.id ? await prisma.user.findUnique({
    where: { id: session.user.id },
    select: { subscriptionTier: true }
  }) : null;

  const userTier = user?.subscriptionTier || 'FREE';

  // Get platforms accessible to user's tier
  const platforms = await prisma.streamingPlatform.findMany({
    where: {
      isActive: true,
      tierAccess: { in: getTierHierarchy(userTier) }
    },
    include: {
      userFavorites: session?.user?.id ? {
        where: { userId: session.user.id }
      } : false,
      _count: { select: { userFavorites: true } }
    }
  });

  return NextResponse.json({ platforms });
}

function getTierHierarchy(tier: string): string[] {
  const hierarchy: Record<string, string[]> = {
    'FREE': ['FREE'],
    'TRIAL': ['FREE', 'PREMIUM'],
    'PREMIUM': ['FREE', 'PREMIUM'],
    'PRO': ['FREE', 'PREMIUM', 'PRO']
  };
  return hierarchy[tier] || ['FREE'];
}
```

`/api/media/streaming-platforms/[platformId]/favorite/route.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth';
import { prisma } from '@/lib/db';

export async function POST(
  req: NextRequest,
  { params }: { params: { platformId: string } }
) {
  const session = await getServerSession(authOptions);
  if (!session?.user?.id) {
    return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
  }

  try {
    await prisma.userFavoriteStreamingPlatform.create({
      data: {
        userId: session.user.id,
        platformId: params.platformId
      }
    });

    // Track engagement
    await prisma.mediaEngagementMetrics.create({
      data: {
        userId: session.user.id,
        contentType: 'STREAMING_PLATFORM',
        contentId: params.platformId,
        engagementType: 'FAVORITE'
      }
    });

    return NextResponse.json({ success: true });
  } catch (error) {
    return NextResponse.json({ error: 'Failed to favorite' }, { status: 500 });
  }
}

export async function DELETE(
  req: NextRequest,
  { params }: { params: { platformId: string } }
) {
  const session = await getServerSession(authOptions);
  if (!session?.user?.id) {
    return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
  }

  try {
    await prisma.userFavoriteStreamingPlatform.delete({
      where: {
        userId_platformId: {
          userId: session.user.id,
          platformId: params.platformId
        }
      }
    });

    return NextResponse.json({ success: true });
  } catch (error) {
    return NextResponse.json({ error: 'Failed to unfavorite' }, { status: 500 });
  }
}
```

TypeScript Types

The Prisma client automatically generates types for all models:

```
import {
  StreamingPlatform,
  TournamentOrganization,
  UserMediaPreferences,
  MediaEngagementMetrics,
  TrialExpirationReminder,
  StreamingPlatformType,
  TournamentOrganizationType,
  MediaContentType,
  MediaEngagementType,
  TrialReminderType
} from '@prisma/client';

// Example: Type-safe function
async function getPlatform(id: string): Promise<StreamingPlatform | null> {
  return await prisma.streamingPlatform.findUnique({
    where: { id }
  });
}
```

Best Practices

1. **Always track engagement** - Use `trackEngagement()` for all user interactions
2. **Check tier access** - Verify user's subscription tier before showing content
3. **Handle errors gracefully** - Use try-catch blocks for database operations
4. **Use transactions** - For operations that span multiple tables
5. **Index efficiently** - Existing indexes are optimized for common queries
6. **Cache frequently accessed data** - Consider Redis for platform lists
7. **Track anonymous users** - Use sessionId when userId is not available
8. **Clean up old data** - Implement retention policies for analytics data

Need Help?

- Prisma Client API: https://www.prisma.io/docs/concepts/components/prisma-client
- Schema Reference: `/home/ubuntu/mindful_champion/nextjs_space/prisma/schema.prisma`
- Research Data: `/home/ubuntu/pickleball_media_research.json`
- Full Summary: `/home/ubuntu/mindful_champion/nextjs_space/SCHEMA_UPDATES_SUMMARY.md`