

Drizzle ORM Migration Guide

Overview

This project has been migrated from Prisma ORM to Drizzle ORM with comprehensive verbose logging. This guide explains the migration process and provides clear examples for converting remaining files.

Migration Status

✅ Completed:

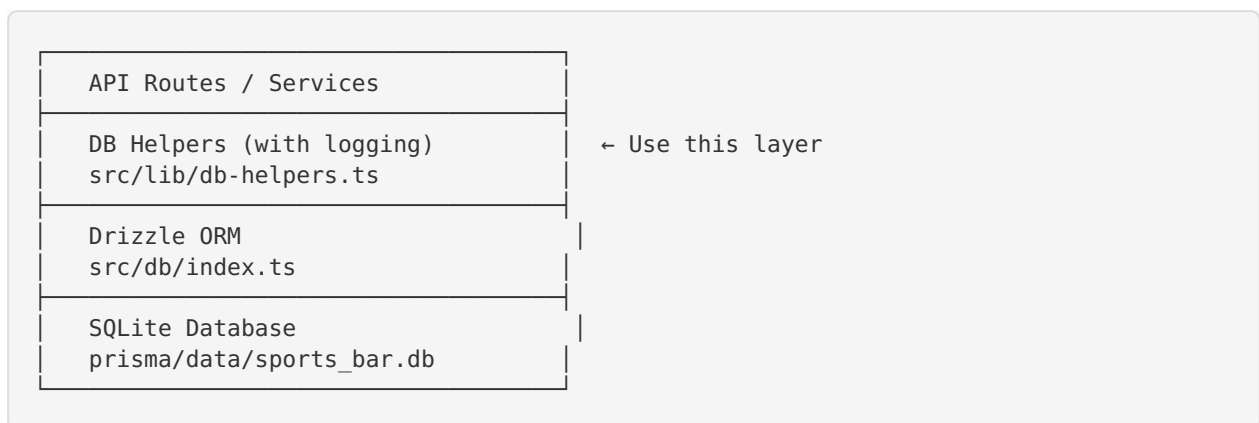
- Drizzle schema created in `src/db/schema.ts`
- Drizzle configuration in `drizzle.config.ts`
- Database connection with logging in `src/db/index.ts`
- Comprehensive logging utility in `src/lib/logger.ts`
- Database helper functions with logging in `src/lib/db-helpers.ts`
- Example API routes migrated:
 - `src/app/api/schedules/route.ts`
 - `src/app/api/home-teams/route.ts`
- Installation scripts updated
- README updated

⚠️ Remaining Work:

- ~104 API routes and service files still using Prisma adapter
- These files need to be migrated to use direct Drizzle with db-helpers

Architecture

Database Layer



Logging System

All database operations, API calls, and system events are logged using the comprehensive logging utility in `src/lib/logger.ts`.

Log Categories:

- `DATABASE` - All database operations (queries, inserts, updates, deletes)
- `API` - API endpoint calls and responses

- ATLAS - Atlas processor TCP communication
- NETWORK - Network requests
- AUTH - Authentication events
- SYSTEM - System startup/shutdown
- CACHE - Cache operations

Migration Pattern

Old Prisma Pattern

```
import { prisma } from '@lib/db';

// GET - List records
export async function GET(request: NextRequest) {
  try {
    const records = await prisma.schedule.findMany({
      where: { enabled: true },
      orderBy: { createdAt: 'desc' }
    });

    return NextResponse.json({ records });
  } catch (error: any) {
    console.error('Error:', error);
    return NextResponse.json(
      { error: 'Failed to fetch records' },
      { status: 500 }
    );
  }
}

// POST - Create record
export async function POST(request: NextRequest) {
  try {
    const body = await request.json();

    const record = await prisma.schedule.create({
      data: {
        name: body.name,
        enabled: body.enabled
      }
    });

    return NextResponse.json({ record }, { status: 201 });
  } catch (error: any) {
    console.error('Error:', error);
    return NextResponse.json(
      { error: 'Failed to create record' },
      { status: 500 }
    );
  }
}
```

New Drizzle Pattern with Logging

```
import { NextRequest, NextResponse } from 'next/server';
import { schema } from '@db';
import { desc, eq } from 'drizzle-orm';
import { logger } from '@lib/logger';
import { findMany, create } from '@lib/db-helpers';

// GET - List records
export async function GET(request: NextRequest) {
  logger.api.request('GET', '/api/schedules');

  try {
    const records = await findMany('schedules', {
      where: eq(schema.schedules.enabled, true),
      orderBy: desc(schema.schedules.createdAt)
    });

    logger.api.response('GET', '/api/schedules', 200, { count: records.length });
    return NextResponse.json({ records });
  } catch (error: any) {
    logger.api.error('GET', '/api/schedules', error);
    return NextResponse.json(
      { error: 'Failed to fetch records', details: error.message },
      { status: 500 }
    );
  }
}

// POST - Create record
export async function POST(request: NextRequest) {
  logger.api.request('POST', '/api/schedules');

  try {
    const body = await request.json();
    logger.debug('Creating record', { data: body });

    const record = await create('schedules', {
      name: body.name,
      enabled: body.enabled
    });

    logger.api.response('POST', '/api/schedules', 201, { id: record.id });
    return NextResponse.json({ record }, { status: 201 });
  } catch (error: any) {
    logger.api.error('POST', '/api/schedules', error);
    return NextResponse.json(
      { error: 'Failed to create record', details: error.message },
      { status: 500 }
    );
  }
}
}
```

Common Conversions

Import Statements

OLD:

```
import { prisma } from '@lib/db';
```

NEW:

```
import { schema } from '@db';
import { logger } from '@lib/logger';
import { findMany, findFirst, findUnique, create, update, deleteRecord, eq, and, or, desc, asc } from '@lib/db-helpers';
```

Find Many**OLD:**

```
const records = await prisma.schedule.findMany({
  where: { enabled: true },
  orderBy: { createdAt: 'desc' },
  take: 10,
  skip: 0
});
```

NEW:

```
const records = await findMany('schedules', {
  where: eq(schema.schedules.enabled, true),
  orderBy: desc(schema.schedules.createdAt),
  limit: 10,
  offset: 0
});
```

Find First/Unique**OLD:**

```
const record = await prisma.schedule.findFirst({
  where: { id: 'some-id' }
});

const unique = await prisma.schedule.findUnique({
  where: { id: 'some-id' }
});
```

NEW:

```
const record = await findFirst('schedules', {
  where: eq(schema.schedules.id, 'some-id')
});

const unique = await findUnique('schedules',
  eq(schema.schedules.id, 'some-id')
);
```

Create**OLD:**

```
const record = await prisma.schedule.create({
  data: {
    name: 'Test',
    enabled: true
  }
});
```

NEW:

```
const record = await create('schedules', {
  name: 'Test',
  enabled: true
});
```

Update**OLD:**

```
const record = await prisma.schedule.update({
  where: { id: 'some-id' },
  data: { enabled: false }
});
```

NEW:

```
const record = await update('schedules',
  eq(schema.schedules.id, 'some-id'),
  { enabled: false }
);
```

Delete**OLD:**

```
await prisma.schedule.delete({
  where: { id: 'some-id' }
});
```

NEW:

```
await deleteRecord('schedules',
  eq(schema.schedules.id, 'some-id')
);
```

Count**OLD:**

```
const count = await prisma.schedule.count({
  where: { enabled: true }
});
```

NEW:

```
const count = await count('schedules',
  eq(schema.schedules.enabled, true)
);
```

Complex Where Clauses**OLD:**

```
const records = await prisma.schedule.findMany({
  where: {
    enabled: true,
    createdAt: {
      gte: new Date('2024-01-01')
    }
  }
});
```

NEW:

```
import { gte } from '@lib/db-helpers';

const records = await findMany('schedules', {
  where: and(
    eq(schema.schedules.enabled, true),
    gte(schema.schedules.createdAt, new Date('2024-01-01').toISOString())
  )
});
```

Multiple Order By**OLD:**

```
const records = await prisma.homeTeam.findMany({
  orderBy: [
    { isPrimary: 'desc' },
    { teamName: 'asc' }
  ]
});
```

NEW:

```
const records = await findMany('homeTeams', {
  orderBy: [
    desc(schema.homeTeams.isPrimary),
    asc(schema.homeTeams.teamName)
  ]
});
```

Table Name Mapping

Prisma model names to Drizzle table names:

Prisma Model	Drizzle Table Name
<code>prisma.schedule</code>	<code>'schedules'</code>
<code>prisma.scheduleLog</code>	<code>'scheduleLogs'</code>
<code>prisma.homeTeam</code>	<code>'homeTeams'</code>
<code>prisma.fireTVDevice</code>	<code>'fireTVDevices'</code>
<code>prisma.matrixConfiguration</code>	<code>'matrixConfigurations'</code>
<code>prisma.matrixInput</code>	<code>'matrixInputs'</code>
<code>prisma.matrixOutput</code>	<code>'matrixOutputs'</code>
<code>prisma.audioProcessor</code>	<code>'audioProcessors'</code>
<code>prisma.audioZone</code>	<code>'audioZones'</code>
<code>prisma.channelPreset</code>	<code>'channelPresets'</code>
<code>prisma.qaEntry</code>	<code>'qaEntries'</code>
<code>prisma.chatSession</code>	<code>'chatSessions'</code>
<code>prisma.document</code>	<code>'documents'</code>

For a complete list, see `src/db/schema.ts`.

Available DB Helper Functions

All functions include comprehensive logging:

- `findMany(tableName, options)` - Find multiple records
- `findFirst(tableName, options)` - Find first matching record
- `findUnique(tableName, where)` - Find unique record
- `create(tableName, data)` - Create single record
- `createMany(tableName, data[])` - Create multiple records
- `update(tableName, where, data)` - Update single record
- `updateMany(tableName, where, data)` - Update multiple records
- `deleteRecord(tableName, where)` - Delete single record
- `deleteMany(tableName, where)` - Delete multiple records
- `count(tableName, where?)` - Count records
- `upsert(tableName, where, createData, updateData)` - Insert or update
- `transaction(callback)` - Execute in transaction

Available Operators

```
import { eq, and, or, desc, asc, inArray, like, gte, lte, gt, lt, ne } from '@lib/db-helpers';
```

- `eq(column, value)` - Equals
- `ne(column, value)` - Not equals
- `gt(column, value)` - Greater than
- `gte(column, value)` - Greater than or equal
- `lt(column, value)` - Less than
- `lte(column, value)` - Less than or equal
- `like(column, pattern)` - Pattern matching
- `inArray(column, values[])` - In array
- `and(...conditions)` - Logical AND
- `or(...conditions)` - Logical OR
- `desc(column)` - Descending order
- `asc(column)` - Ascending order

Logging Best Practices

API Endpoints

Always log API requests and responses:

```
export async function GET(request: NextRequest) {
  logger.api.request('GET', '/api/endpoint');

  try {
    // ... your code ...
    logger.api.response('GET', '/api/endpoint', 200, { count: results.length });
    return NextResponse.json({ results });
  } catch (error: any) {
    logger.api.error('GET', '/api/endpoint', error);
    return NextResponse.json({ error: 'Failed' }, { status: 500 });
  }
}
```

Database Operations

Use the db-helper functions which include automatic logging, or log manually:

```
// Automatic logging via helpers
const records = await findMany('schedules', { ... });

// Or log manually for custom queries
logger.database.query('customQuery', 'schedules', { params });
const result = await db.select().from(schema.schedules).where(...);
logger.database.success('customQuery', 'schedules', result);
```


Atlas Communication

```
logger.atlas.connect(ipAddress, port);
logger.atlas.command('setVolume', { zone: 1, volume: 50 });
logger.atlas.response('setVolume', response);
```

Database Commands

Development

```
# Generate migrations
npm run db:generate

# Push schema changes to database
npm run db:push

# Open Drizzle Studio (database GUI)
npm run db:studio
```

Schema Location

The database schema is now defined in TypeScript:

- **Schema file:** `src/db/schema.ts`
- **Database file:** `prisma/data/sports_bar.db` (unchanged)
- **Migrations:** `drizzle/` directory

Testing After Migration

After migrating a file:

1. Build the project:

```
bash
npm run build
```

2. Test the API endpoint:

```
bash
curl http://localhost:3000/api/your-endpoint
```

3. Check logs:

- Verify database operations are logged
- Verify API requests/responses are logged
- Check for any errors

4. Test in browser:

- Use the application UI to test functionality
- Open browser console to see logs (if applicable)

Migration Checklist

For each file you migrate:

- ☐ Update import statements
- ☐ Convert all Prisma queries to Drizzle with db-helpers

- [] Add API request/response logging
- [] Add error logging
- [] Test the endpoint
- [] Verify logs are appearing
- [] Commit changes

Examples to Study

Completed Migrations:

1. `src/app/api/schedules/route.ts` - Complete CRUD with logging
2. `src/app/api/home-teams/route.ts` - Multiple orderBy with logging
3. `src/db/index.ts` - Database connection with logging
4. `src/lib/logger.ts` - Comprehensive logging utility
5. `src/lib/db-helpers.ts` - Database helpers with logging

Need Help?

- Review the completed example files
- Check `src/db/schema.ts` for table structures
- See `src/lib/db-helpers.ts` for available functions
- Consult Drizzle ORM documentation: <https://orm.drizzle.team/>

Backward Compatibility







The Prisma compatibility adapter (`src/db/prisma-adapter.ts`) is kept temporarily for files that haven't been migrated yet. However:

 **This adapter is DEPRECATED and will be removed**

All new code should use direct Drizzle ORM with the db-helpers.

Performance Notes

Drizzle ORM benefits:

-  Lightweight (no generation step needed)
-  Type-safe SQL queries
-  Better performance than Prisma
-  Full control over queries
-  Comprehensive logging built in
-  Smaller bundle size

Support

If you encounter issues during migration:

1. Check this guide for the correct pattern
2. Review completed example files
3. Verify your imports and table names
4. Check the logs for detailed error messages
5. Test incrementally (one file at a time)