Performance Optimization Guide

1. Database Optimization

Indexes

Chúng tôi đã tạo các indexes quan trọng cho:

- **Submission**: `status`, `authorId`, `createdAt`, `title`, `abstract_vi`
- **Article**: `issueId`, `publishedAt`, `categoryId`, `title`
- **Review**: `status`, `reviewerId`, `submissionId`, `deadline`
- **Issue**: `status`, `year`, `volumeId`
- **AuditLog**: `actorId`, `action`, `createdAt`, `ipAddress`

Maintenance Tasks

```
# 1. Analyze tables (cập nhật statistics)
psql $DATABASE_URL -f scripts/optimize-db.sql

# 2. Vacuum định kỳ (mỗi tuần)
psql $DATABASE_URL -c "VACUUM ANALYZE;"

# 3. Reindex khi cần (mỗi tháng)
psql $DATABASE_URL -c "REINDEX DATABASE tapchi_hcqs;"
```

Query Optimization

❌ **Bad Practice:**

```
// Không dùng select all
const users = await prisma.user.findMany();

// Không fetch toàn bộ relations
const submission = await prisma.submission.findUnique({
  where: { id },
  include: { author: true, reviews: true, versions: true }
});
```

✅ **Best Practice:**

```
// Chỉ select fields cần thiết
const users = await prisma.user.findMany({
  select: { id: true, fullName: true, email: true },
  take: 50,
});

// Fetch chỉ relations cần thiết
const submission = await prisma.submission.findUnique({
  where: { id },
  select: {
    id: true,
    title: true,
    author: { select: { id: true, fullName: true } },
  },
});
```

2. API Route Optimization

Caching Strategy

```
// Cache static data
import { unstable_cache } from 'next/cache';

export const getCategories = unstable_cache(
  async () => {
    return await prisma.category.findMany();
  },
  ['categories'],
  { revalidate: 3600 } // 1 hour
);
```

Rate Limiting

```
import { rateLimit, RateLimitPresets } from '@/lib/rate-limit';

export async function POST(request: NextRequest) {
  // Apply rate limiting
  const rateLimitResult = rateLimit(RateLimitPresets.standard)(request);
  if (rateLimitResult) return rateLimitResult;

  // Process request...
}
```

Pagination

```
// Always paginate large datasets
const page = parseInt(searchParams.get('page') || '1');
const limit = 50;
const skip = (page - 1) * limit;

const [items, total] = await Promise.all([
  prisma.submission.findMany({
    skip,
    take: limit,
    orderBy: { createdAt: 'desc' },
  }),
  prisma.submission.count(),
]);
```

3. Frontend Optimization

Lazy Loading Components

```
import dynamic from 'next/dynamic';

// Lazy load heavy components
const ChartComponent = dynamic(() => import('./chart'), {
  loading: () => <Skeleton />,
  ssr: false,
});
```

Image Optimization

```
import Image from 'next/image';

// Always use Next.js Image component
<Image
  src={coverImage}
  alt={title}
  width={300}
  height={400}
  placeholder="blur"
  priority={isFeatured}
/>
```

Data Fetching

```
// Use React Query for client-side caching
import { useQuery } from '@tanstack/react-query';

const { data, isLoading } = useQuery({
  queryKey: ['submissions', page],
  queryFn: () => fetchSubmissions(page),
  staleTime: 5 * 60 * 1000, // 5 minutes
});
```

4. File Upload Optimization

S3 Direct Upload

```
// ✅ Upload directly to S3 (not through server)
const { uploadUrl, cloud_storage_path } = await getPresignedUrl();
await fetch(uploadUrl, {
  method: 'PUT',
  body: file,
  headers: { 'Content-Type': file.type },
});
```

Image Resizing

```
// Consider resizing images before upload
import sharp from 'sharp';

const resized = await sharp(buffer)
  .resize(1200, 1200, { fit: 'inside' })
  .jpeg({ quality: 85 })
  .toBuffer();
```

5. Monitoring & Metrics

Database Monitoring

```
-- Check table sizes
SELECT * FROM table_sizes;

-- Check slow queries
SELECT * FROM slow_queries;

-- Check index usage
SELECT * FROM index_usage WHERE index_scans = 0;

-- Check active connections
SELECT count(*) FROM pg_stat_activity;
```

Application Monitoring

```
# Check application health
curl http://localhost:3000/api/health

# Check memory usage
docker stats tapchi-app

# Check logs for errors
docker logs tapchi-app --tail 100 | grep ERROR
```

6. Performance Benchmarks

Target Metrics

- **Page Load Time**: < 2 seconds

- **API Response Time**: < 500ms (p95)
- **Database Query Time**: < 100ms (p95)
- **Time to Interactive (TTI)**: < 3 seconds
- **Largest Contentful Paint (LCP)**: < 2.5 seconds

Testing

```
# Load testing with k6
k6 run load-test.js

# Database query analysis
psql $DATABASE_URL -c "EXPLAIN ANALYZE SELECT ..."
```

7. Production Checklist

- [ ] All critical queries have proper indexes
- [ ] Rate limiting enabled on all API routes
- [ ] Caching strategy implemented
- [ ] Image optimization enabled
- [ ] Gzip compression enabled (Nginx)
- [ ] Database maintenance scheduled (weekly VACUUM)
- [ ] Monitoring dashboards configured
- [ ] Error tracking enabled
- [ ] Log rotation configured
- [ ] Backup verification tested

8. Common Performance Issues

Issue: Slow Dashboard Loading

Cause: Fetching too much data at once

Solution:

```
// Before: Fetch everything
const submissions = await prisma.submission.findMany({
  include: { author: true, reviews: true, versions: true }
});

// After: Paginate and select only needed fields
const submissions = await prisma.submission.findMany({
  select: {
    id: true,
    title: true,
    status: true,
    author: { select: { fullName: true } },
  },
  take: 20,
  orderBy: { createdAt: 'desc' },
});
```

Issue: High Database CPU

Cause: Missing indexes or inefficient queries

Solution:

```
# Identify slow queries
psql $DATABASE_URL -c "SELECT * FROM slow_queries;"

# Check missing indexes
psql $DATABASE_URL -f scripts/optimize-db.sql
```

Issue: Memory Leaks

Cause: Not closing database connections

Solution:

```
// Always use Prisma's connection pooling
// Don't create new PrismaClient instances in API routes
import { prisma } from '@/lib/prisma'; // ✅ Reuse singleton
```

9. Scaling Considerations

Horizontal Scaling

- Use Redis for session storage (not in-memory)
- Use external file storage (S3, not local filesystem)
- Use database read replicas for heavy read operations
- Use load balancer (Nginx) for multiple app instances

Vertical Scaling

- Increase database connection pool size
- Increase Node.js memory limit: `NODE_OPTIONS=--max-old-space-size=4096`
- Increase worker processes in Nginx

Resources

- Prisma Performance Guide (https://www.prisma.io/docs/guides/performance-and-optimization)
- Next.js Performance (https://nextjs.org/docs/app/building-your-application/optimizing)
- PostgreSQL Performance Tips (https://wiki.postgresql.org/wiki/Performance_Optimization)